Identification and Self-Tuning Control of Dynamic Systems

by

Ali Yurdun Orbak

B.S., İstanbul Technical University (1992)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author .....
Department of Mechanical Engineering
May 12, 1995

Certified by .....
Kamal Youcef-Toumi
Associate Professor
Thesis Supervisor

Accepted by .....
Ain A. Sonin
Chairman, Departmental Committee on Graduate Students

Identification and Self-Tuning Control of Dynamic Systems

by

Ali Yurdun Orbak

Submitted to the Department of Mechanical Engineering
on May 12, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Self-tuning is a direct digital control technique in which the controller settings are automatically modified to compensate for changes in the characteristics of the controlled process. An algorithm has the self-tuning property, if the controller parameters tend to those corresponding values of an exactly known plant model, as the number of samples approaches infinity. This thesis presents the types and the basic formulation of self-tuners and then explains the algorithm for Minimum Variance Control in general. It also includes some identification techniques (such as the Recursive Parameter Estimation method) that are most popular for self-tuners. Minimum variance control and identification routines were prepared in Matlab^{TM} and used successfully in several applications. The scripts developed in this thesis were prepared in toolbox like functions; where a user can easily use them for any system. Different methods of self-tuning can also be included in these scripts without much difficulty.

Thesis Supervisor: Kamal Youcef-Toumi
Title: Associate Professor

Acknowledgments

First, I would like to thank Prof. Kamal Youcef-Toumi for his guidance and support throughout my master's degree. I have learned much insight about controls during our invaluable discussions. Kamal, it has been a real pleasure working with you! I also want to thank to Prof. Can Özsoy, my advisor in Turkey, for his valuable comments on my work.

Thanks to Cüneyt Yılmaz, my colleague in Turkey, who helped me with the programming in this work. It would not be possible to construct almost perfect programs without his valuable ideas and debugging.

Thanks to my friends at MIT who were always there when I needed them.

...And finally I would like to THANK my parents and my brother, for their love, never ending support and encouragement. Thanks for helping me go through my outside Turkey experience and MIT; these years would not have been the same without your love.

To my mother Nurten, my father Günhan,
and my brother İlkün.

Contents

List of Figures

Chapter 1

Introduction

The control system theorists and practitioners have been dealing with the adaptive control of systems for more than a quarter of a century. This class of control systems arose from a desire and need for improved performance of increasingly complex engineering systems with large uncertainties [2]. The matter is especially important in systems with many unknown parameters which are also changing with time.

The *tuning* problem is one reason for using adaptive control [2, 4]. It is a very well known fact that many processes can be regulated satisfactorily with PI or PID controllers. And it is easy to tune a PI regulator with only two parameters to adjust. However, if the problem in hand is an installation with several hundred regulators, it is a difficult task to keep all the regulators well tuned [2]. On the other hand, even a PID regulator with three or four parameters is not always easy to tune, especially if the process dynamics is slow.

There are several ways to tune regulators. One possibility is to develop a mathematical model (can be from physical modelling or system identification) for the process and disturbances, and to derive the regulator parameters from some control design. The drawback of this method is its need for engineer with skills in modelling, system identification and control design, and also it can be a time consuming procedure. At this point, the self-tuning regulator can be regarded as a convenient way to combine system identification and control design [2]. In fact its name comes from such applications.

In a broad sense, the purpose of self-tuning control is to control systems with unknown but constant or slowly varying parameters. And the basic idea can be described as follows; "start with a design method that gives adequate results if the parameters of models for the dynamics of the process and its environment are known. When the parameters are unknown, replace them by estimates obtained from a recursive parameter estimator" [4]. There are many available methods for designing control systems, so there are at least as many ways to design self-tuning regulators. The next sections give a brief summary of the commonly used types of self-tuning control together with their advantages and disadvantages[1].

1.1 Linear Quadratic Gaussian Self-tuners

Control engineers know that, LQG theory is a success in the control of known plants. Although this is the case, it has not often been applied to self-tuning systems. Åström and Wittenmark [5] have discussed the use of explicit LQG regulators and have developed a microcomputer based system[2]. Also an explicit LQG regulator that uses both state-space and input-output models has been described in the literature. Some of the research on state-space based approaches has the advantage that numerical problems involved in iterating the Riccati equation are avoided [31].

The ability of implicit self-tuning algorithms have been recognized by Åström and Wittenmark, and Clarke and Gawthrop. The LQG controllers have good stability characteristics in comparison with the Minimum Variance and Generalized Minimum Variance controllers employed with these authors [31]. However, especially in the multivariable case, the LQG controller is more complicated to calculate. Square root and fast algorithms for solving the LQG control problem in discrete time systems have been established together with the computational complexity, that is the arithmetic operations and storage requirements, in previous years. And, in fact, the best form

[1]The Minimum Variance Self-tuning control will not be included in these sections since it is the focus of this work.

[2]This work can be found in: Zhao-Ying and Åström. A microcomputer implementation of LQG self-tuner. Research report LUTFDL/(TFRT-7226)/1-052/1981, 1981, Lund Institute of Technology.

of LQG self-tuning controller will be determined by comparison of such factors [31].

1.2 Self-tuning Proportional Integral Derivative Controllers

One of the advantages of the well-known PID controller is that, it is sufficiently flexible for many control applications [31]. Generally the process in closed loop and the controller is in tune, but sometimes the tuning may be quite time consuming and this time one can be interested in tuning them automatically. The idea of self-tuning regulators has been introduced to simplify the tuning of industrial controllers [31]. Those regulators also have parameters to tune. The new parameters should, however, be easier to choose than the parameters in conventional controllers. It is desired to have as few tuning parameters as possible. It is also obvious that the user should provide the controller with information about the desired specifications (i.e. percent overshoot, settling time, etc.). So the self-tuning PID controller also has parameters that are related to the performance of the closed loop behavior [31].

1.3 Hybrid Self-tuning Control

Self-tuning controllers have traditionally been developed within a discrete time framework. However, many model reference methods have been built in continuous time. For this reason, in order to combine the advantages of both time domains, hybrid methods have been suggested [13, 31]. This method has these advantages when compared with the purely discrete time counterpart[3];

a) The stability of the algorithm depends on the zeros of the system expressed in a continuous time form. On the other hand, discrete time methods depend on the zeros of the discrete time impulse equivalent transfer function. And we know

[3]This background information part has been taken from [31].

that the zeros of the discrete time form may lie outside the unit circle even if the zeros of the continuous time form lie within the left half plane.

b) The estimated controller parameters are in a continuous time form. This is an advantage in numerical calculations and the parameters are more meaningful than the discrete time counterparts.

c) The method leads to self-tuning PI and PID controllers.

And the method has the following advantages when compared with its continuous time counterpart;

a) The algorithm translates directly into a flexible digital implementation, consistent with a modern distributed control system.

b) Several applications and preliminary results suggested that the influence of unmodelled high frequency components is less severe than when using a continuous time algorithm.

1.4 Pole Placement Self-tuning Control

From literature discussions, we learn that the minimum variance related self-tuning algorithms (Åström and Wittenmark 1973, Clarke and Gawthrop 1975) are based on an optimal control concept whereby the control or regulation strategy is to minimize a cost function. On the other hand, the philosophy in pole placement (or assignment) self-tuning[4] is to specify the control strategy in terms of a desired closed loop pole set. Then the aim of the self-tuning algorithm is to position the actual closed loop poles at desired locations [31].

There are a few drawbacks of this method. The obvious drawback is its nonoptimality [31] compared with Minimum Variance Control. According to Singh, specifically, the variability of the regulated output under pole placement may be significantly

[4]Wellstead P. E., Prager D. L., Zanker P. M. Pole assignment self-tuning regulator. *Proc. Inst. Electr. Eng.* 126:781–787,1979.

reduced by adjoining an optimization section to the basic pole placement algorithm. In addition, the transient response of a pole placement self-tuner will be influenced by the system zeros [31].

From an operational viewpoint, pole placement involves more on line computation than do the minimum variance methods. As an example, there is usually an identity to solve in pole placement [31].

Besides these types, self-tuning control can also be used as self-tuning feedforward control or self-tuning on-off control. But the applications of these types are very few. Because of this reason, they will not be covered in this thesis.

1.5 Outline Of This Work

This work is principally about the basic formulation of self-tuners and their applications. The self-tuning algorithm discussed here is the Minimum Variance Control. It also includes important identification techniques such as Least Squares technique that are of great importance in self-tuning control.

This thesis is organized as follows: The second chapter first explains the basic formulation of dynamic systems in discrete time. Then it presents the basic technique of Least Squares and the formulation of Minimum Variance Control along with the self-tuning regulator (STR). This chapter ends with the explanations and comparisons of common types of self-tuners (i.e. implicit and explicit algorithms).

The third chapter deals with the formulation of the control algorithm for simulation purposes. It also discusses several applications of the Minimum Variance self-tuning algorithm to dynamic systems. Besides, this chapter also includes some identification topics such as the system identification toolbox of Matlab^{TM}. Then, in this chapter an ARX[5] modelling script is introduced together with simulations.

The last chapter gives the results of this work and makes suggestions and recommendations for future work.

In the appendices the programming routines (scripts) of the simulations are given.

[5] Auto Regressive with eXogeneous inputs.

These routines are also pretty useful for those who want to start learning self-tuning control. It also has some advanced routines, which the author thinks, will be useful to the researchers at any level. The scripts given were prepared as a toolbox, in the sense that they can easily be adapted to any plant. Other types of self-tuning algorithms can also be added to these scripts without much difficulty.

Chapter 2

Introduction to Self-tuning Controllers

Adaptive control is an important field of research in control engineering. It is also of increasing practical importance, because adaptive control techniques are being used more and more in industrial control systems. In particular, self-tuning controllers and self-tuning regulators (STR) represent an important class with increasing theoretical and practical interests. The original concepts of this type of control were conceived by R.E. Kalman in the early 60's. It now forms an important branch of adaptive control systems in particular, and of control systems in general.

As mentioned before, the objective of the STR is to control systems with unknown, constant or slowly varying parameters. Consequently, the theoretical interest is centered around stability, performance, and convergence of the recursive algorithms (usually Recursive Least Squares) involved. On the other hand, the practical interest stems from its potential uses, both as a method for controlling time varying and nonlinear plants over a wide range of operating points, and for dealing with batch problems over a wide range of operating points where the plant or materials involved may vary over successive batches [15]. However, in the past, successful implementations have been restricted primarily to applications in the pulp and paper, chemical and other resource based industries where the process dynamics are significantly slower than the STR algorithm. But in these days, merging LQG/LTR (Linear Quadratic

Gaussian/Loop Transfer Recovery) control with STR, made it faster, more practical and more robust.

According to Åström, the STR is based on the idea of separating the estimation of unknown parameters from the design of the controller. By using the recursive estimation methods, it is possible to estimate the unknown parameters [6].

For this purpose, the most straightforward approach is to estimate the parameters of the transfer function of the process and the disturbances [6]. This gives an *indirect adaptive algorithm.* An example is a controller based on least squares estimation and minimum variance control, in which the uncertainties of the estimates were considered. This was published by Wieslander and Wittenmark in 1971.

It is often possible to use reparameterization on the model of the regulator to estimate regulator parameters directly. This time, it is called a *direct adaptive algorithm.*

In self-tuning context, indirect methods have often been termed *explicit self-tuning control*, since the process parameters have been estimated. Direct updating of the regulator parameters is called as *implicit self-tuning control* [6].

Having presented a brief review of self-tuning controllers, their basic formulation is outlined in the next section.

2.1 System Models

For the sake of simplicity in describing STR, the typical continuous time, single input single output (SISO) system is chosen. In this case, a system can typically be described by the following equation from which one can obtain the transfer function:

$$y(t) = \frac{B(s)}{A(s)} u(t - \Delta) + d(t) \tag{2.1}$$

$$G(s) = e^{-s\Delta} \frac{B(s)}{A(s)} \tag{2.2}$$

Here $A(s)$ and $B(s)$ are polynomials in the differential operator s and Δ is the dead-time[1] before the plant input $u(t)$ affects to output $y(t)$. $d(t)$ is a general disturbance. If one considers the stochastic signal for the disturbance[2] ($d(t) = G_2(s)\,\xi(t)$) where $\xi(t)$ is white noise, the following equation for the system is obtained:

$$y(t) = \frac{B(s)}{A(s)}\,u(t-\Delta) + \frac{C(s)}{A(s)}\,\xi(t) \tag{2.3}$$

Here the only restriction on the polynomial $C(s)$ is that, no root lies in right half plane or on imaginary axis. As the self-tuners are implemented digitally, the system equations need to be converted to discrete form by using a zero-order hold (ZOH) block, for example. The process model, as seen by the controller after the ZOH has been inserted, is as shown in Figure 2-1. If the sample interval is denoted by h, and using $z = e^{s\,h}$, one obtains:

$$G(z^{-1}) = z^{-k}\frac{(b_0 + b_1 z^{-1} + \cdots + b_n z^{-n})}{(1 + a_1 z^{-1} + \cdots + a_n z^{-n})} = z^{-k}\frac{\bar{B}(z^{-1})}{\bar{A}(z^{-1})} \tag{2.4}$$

and

$$y(t) + \sum_{i=1}^{n} a_i\,y(t - i\,h) = \sum_{i=0}^{n} b_i\,u(t - i\,h - k\,h) \tag{2.5}$$

in the difference equation form ($t = i\,h$).

As a result the system equation becomes:

$$y(t) = \frac{\bar{B}(z^{-1})}{\bar{A}(z^{-1})}\,u(t - k\,h) + \frac{\bar{C}(z^{-1})}{\bar{A}(z^{-1})}\,\xi(t) \tag{2.6}$$

Here the $\xi(t)$ is an uncorrelated sequence with variance σ^2. It is also assumed that $b_0 \neq 0$ so that $k > 1$, $a_0 = c_0 = 1$ and all the polynomials are of order n, and all roots of polynomial $\bar{C}(z^{-1})$ lie within the unit circle. Also note that the dead-time of k samples is $INT(\Delta/h) + 1$. From now on, since z is interpreted as the forward shift operator, polynomials such as $\bar{A}(z^{-1})$ will be referred simply as A for convenience[3].

[1] $\Delta = (k-1)h + \delta, 0 < \delta \leq h$, δ is the "fractional delay". For further discussion refer to [13].

[2] Transfer function $G_2(s)$ gives special density to $d(t)$ as $G_2(j\omega)\,G_2(-j\omega)$.

[3] Also in presenting the self-tuning algorithms the plant model will always be assumed in the

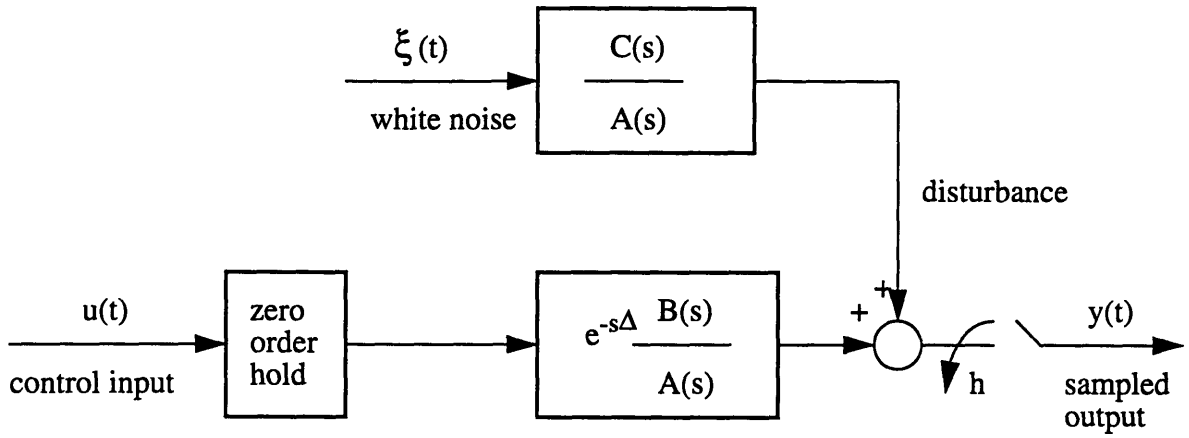

Figure 2-1: Open loop structure of the plant model in discrete time

The above result can also be written in difference equation form as:

$$y(t) + \sum_{i=1}^{n} a_i\, y(t - i\,h) = \sum_{i=0}^{n} b_i\, y(t - i\,h - k\,h) + \xi(t) + \sum_{i=1}^{n} c_i\, \xi(t - i\,h) \tag{2.7}$$

Note that, many self tuning strategies, particularly implicit methods, are based on predictive control designs, where the prediction horizon is the system delay k [13]. (If k is unknown, one procedure is to assume some minimum value (such as 1) and to have an extended B polynomial[4] for which the leading $k-1$ terms should be zero, giving the *Deterministic Autoregressive and Moving Average* (DARMA) form [14].) So now let's examine the predictive models.

The system described by equation (2.6) can also be rewritten as:

$$y(t + k\,h) = \frac{B}{A}\, u(t) + \frac{C}{A}\, \xi(t + k\,h) \tag{2.8}$$

Here $\frac{C}{A}$ can be resolved in the following identity which is explained below [13]:

$$\frac{C}{A} = E(z^{-1}) + z^{-k}\, \frac{F(z^{-1})}{A} \tag{2.9}$$

standard discrete-time form. So from now on the bar can also be dropped.

[4] B polynomial is in the form: $B(z^{-1}) = b_0 + b_1 z^{-1} + \cdots + b_{k-1} z^{-(k-1)} + \cdots + b_n z^{-n}$.

where the degree of the polynomial E (it is in the form $E(z^{-1}) = 1 + e_1 z^{-1} + \cdots + e_{k-1} z^{-(k-1)}$) is $k-1$ with $E(0) = 1$ and the degree of polynomial F is $n-1$. This formulation can be rewritten by rearranging equation (2.8):

$$A\,y(t+k\,h) = B\,u(t) + C\,\xi(t+k\,h) \tag{2.10}$$

Now if we multiply both sides of this equation by E and define a *Diaphontine* equation(algebraic matrix polynomial equation) as:

$$C = E\,A + z^{-k}\,F \tag{2.11}$$

and by the help of this equation we obtain:

$$E\,A\,y(t+k\,h) = E\,B\,u(t) + E\,C\,\xi(t+k\,h) \tag{2.12}$$

$$C\,y(t+k\,h) - F\,y(t) = G\,u(t) + E\,C\,\xi(t+k\,h)\ ;\ G(z^{-1}) = E\,B \tag{2.13}$$

$$y(t+k\,h) = \frac{F\,y(t) + G\,u(t)}{C} + E\,\xi(t+k\,h) \tag{2.14}$$

If the error is defined as $\tilde{y}(t+k\,h|t)$ and the optimum prediction as $y^*(t+k\,h|t)$, one can write $y = y^* + \tilde{y}$ and as a result one obtains[5]:

$$C\,y^*(t+k\,h|t) = F\,y(t) + G\,u(t) \tag{2.15}$$

for the optimal predictor of $y(t+k\,h)$ with error:

$$\tilde{y}(t+k\,h|t) = E\,\xi(t+k\,h) \tag{2.16}$$

Here the prediction accuracy can be measured by the variance of the error $\tilde{y}$, $Var(\tilde{y}) = \sigma^2(1 + e_1^2 + \cdots + e_{k-1}^2)$. This variance increases with k. For further discussion on predictive control and system models the reader can refer to [13, 46, 66, 14].

[5]This notation means that, $y^*(t+k\,h|t)$ is the best estimate (prediction) of $y(t+k\,h)$ based on data up to time t. For further information the reader should refer to [63].

Now, in order to proceed with the formulation of self-tuners, the basics of parameter estimation algorithms need to be reviewed. The next section will introduce the basic formulation of the Least Squares method.

2.2 Recursive Parameter Estimation

According to Clarke, in regression analysis (Plackett, 1960), an observation ("output") is hypothesized to be a linear combination of "inputs", and a set of observations is used to estimate the weighting on each variable such that some fitting criterion is *optimized* [13]. The most commonly used criterion is the "least squares method", where the criterion chooses the model parameters such that the sum-of-squares of the errors between the model outputs and the observations is minimized. Consider the system in equation (2.6). This system can be written in a different form for estimation purposes. The equations of the model that is linear in the parameters are:

$$\phi(t) = \theta_1 x_1(t) + \theta_2 x_2(t) + \cdots + \theta_n x_n(t) + \varepsilon(t) \tag{2.17}$$

or in vector notation

$$\phi(t) = \mathbf{x}^T(t)\,\boldsymbol{\theta} + \varepsilon(t) \tag{2.18}$$

where

- $\phi(t)$ is the "output" observation,
- $\varepsilon(t)$ is the error[6],
- $\mathbf{x}(t)$ is the regression vector of measured input/output data in the form:

$$\mathbf{x}^T(t) = [y(t-1), \cdots, y(t-n), u(t-k), \cdots, u(t-k-n), \varepsilon(t-1), \cdots, \varepsilon(t-n)]$$

[6]Error is statistically independent of the elements $x_i(t)$. These values $(\varepsilon(t-1), \cdots, \varepsilon(t-n))$ are unknown since they are the part of the unobservable white-noise disturbance. For further discussion, the reader should refer to [63].

- $\boldsymbol{\theta}$ is the unknown parameter vector (whose elements are assumed constant):

$$\boldsymbol{\theta}^T = [-a_1, \cdots, -a_n, b_0, \cdots, b_n, c_1, \cdots, c_n]$$

For this discussion assume that $C(z^{-1}) \equiv 0$ so that $c_1, c_2, \cdots, c_n$ are zero and unknown noise is not in $\mathbf{x}(t)$. Then, for estimation purposes one can write:

$$\phi(t) = \mathbf{x}^T(t)\,\hat{\boldsymbol{\theta}} + e(t) \tag{2.19}$$

where $\hat{\boldsymbol{\theta}}$ is a vector of adjustable model parameters, and $e(t)$ is the corresponding modelling error at time t. In a similar notation, one can show that:

$$e(t) = \varepsilon(t) + \mathbf{x}^T(t)\,(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \tag{2.20}$$

So, if the modelling errors are minimized, the only error will be the white-noise that corrupts the output data.

Now, suppose that the system runs for sufficient time to form N consecutive data vectors. Then we have:

$$\begin{bmatrix} \phi(1) \\ \phi(2) \\ \vdots \\ \phi(N) \end{bmatrix} = \begin{bmatrix} \mathbf{x}^T(1) \\ \mathbf{x}^T(2) \\ \vdots \\ \mathbf{x}^T(N) \end{bmatrix} \hat{\boldsymbol{\theta}} + \begin{bmatrix} e(1) \\ e(2) \\ \vdots \\ e(N) \end{bmatrix} \tag{2.21}$$

If this equation is collected and stacked as one equation, one obtains:

$$\boldsymbol{\phi}_N = \mathbf{X}_N\,\hat{\boldsymbol{\theta}} + \mathbf{e}_N \tag{2.22}$$

where

$$\boldsymbol{\phi}_N^T = [\phi(1), \phi(2), \cdots, \phi(N)]$$

and

$$\mathbf{X}_N = \begin{bmatrix} \mathbf{x}^T(1) \\ \mathbf{x}^T(2) \\ \vdots \\ \mathbf{x}^T(N) \end{bmatrix}$$

and $\mathbf{e}_N$ is the model error vector.

If the definition of our loss function is:

$$L = \sum_{i=1}^{N} e_i^2(t) = \mathbf{e}^T \mathbf{e} \tag{2.23}$$

where the model error $\mathbf{e}$ is defined as:

$$\mathbf{e} = \boldsymbol{\phi}_N - \mathbf{X}_N \hat{\boldsymbol{\theta}} \tag{2.24}$$

From these equations one can obtain[7]:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}_N^T \mathbf{X}_N)^{-1} \mathbf{X}_N^T \boldsymbol{\phi}_N \tag{2.25}$$

This equation is referred to as "normal equation of least squares" [13, 14]. Stacking equation (2.18) for $t = 1, \cdots, N$ and substituting in the above equation:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= (\mathbf{X}_N^T \mathbf{X}_N)^{-1} \mathbf{X}_N^T (\mathbf{X}_N \boldsymbol{\theta} + \boldsymbol{\varepsilon}_N) \\ &= \boldsymbol{\theta} + (\mathbf{X}_N^T \mathbf{X}_N)^{-1} \mathbf{X}_N^T \boldsymbol{\varepsilon}_N \end{aligned} \tag{2.26}$$

Remember that here noise has zero mean. If the mean is not zero, we can define $\mathbf{x}_{n+1}(t)$ as 1 and estimate the mean of ε. As a result $E\{\hat{\boldsymbol{\theta}}\} = \boldsymbol{\theta}$, and we have unbiased estimates.

As one knows from the nature of self-tuning, it is useful to make parameter estimation scheme iterative, in order to allow the estimated model to be updated at each sample interval. Now for recursive scheme one could do the following:

[7]Writing $L = (\boldsymbol{\phi}_N - \mathbf{X}_N \hat{\boldsymbol{\theta}})^T (\boldsymbol{\phi}_N - \mathbf{X}_N \hat{\boldsymbol{\theta}})$ and calculating $\frac{\partial L}{\partial \hat{\boldsymbol{\theta}}} = 0$.

Noting that the dimension of $\mathbf{X}^T\mathbf{X}$ does not increase with N, define:

$$\mathbf{S}(t) = \mathbf{X}(t)^T \mathbf{X}(t) \tag{2.27}$$

where $\mathbf{X}(t)$ is the matrix of known data acquired up to time t.

As $\hat{\boldsymbol{\theta}}$ is the nth order vector of estimates, using all data up to time t, equation(2.25) becomes:

$$\begin{aligned} \hat{\boldsymbol{\theta}}(t) &= \mathbf{S}(t)^{-1} \left[\mathbf{X}(t-1)^T \boldsymbol{\phi}(t-1) + \mathbf{x}(t)\, \phi(t)\right] \\ &= \mathbf{S}(t)^{-1} \left[\mathbf{S}(t-1)\, \hat{\boldsymbol{\theta}}(t-1) + \mathbf{x}(t)\, \phi(t)\right] \end{aligned} \tag{2.28}$$

but we know that $\mathbf{S}(t) = \mathbf{S}(t-1) + \mathbf{x}(t)\, \mathbf{x}^T(t)$, so:

$$\begin{aligned} \hat{\boldsymbol{\theta}}(t) &= \mathbf{S}(t)^{-1} \left[\mathbf{S}(t)\, \hat{\boldsymbol{\theta}}(t-1) - \mathbf{x}(t)\, \mathbf{x}^T(t)\, \hat{\boldsymbol{\theta}}(t-1) + \mathbf{x}(t)\, \phi(t)\right] \\ &= \hat{\boldsymbol{\theta}}(t-1) + \mathbf{S}(t)^{-1}\, \mathbf{x}(t) \left[\phi(t) - \mathbf{x}^T(t)\, \hat{\boldsymbol{\theta}}(t-1)\right] \\ \hat{\boldsymbol{\theta}}(t) &= \hat{\boldsymbol{\theta}}(t-1) + \mathbf{K}(t) \left[\phi(t) - \mathbf{x}^T(t)\, \hat{\boldsymbol{\theta}}(t-1)\right] \end{aligned} \tag{2.29}$$

where $\mathbf{K}(t)$ is called as the *Kalman* gain. This equation can also be interpreted as [13]:

$$\begin{pmatrix} new \\ estimate \end{pmatrix} = \begin{pmatrix} old \\ estimate \end{pmatrix} + gain \times \begin{pmatrix} Prediction\ error \\ of\ old\ model \end{pmatrix}$$

If one defines $\mathbf{P}(t)$ as the inverse of $\mathbf{S}(t)$ (in order to get rid of finding an inverse of a matrix in the algorithm, matrix inversion lemma can be used (Appendix C)), one obtains:

$$\mathbf{P}(t) = (\mathbf{S}(t-1) + \mathbf{x}\,\mathbf{x}^T)^{-1} = \mathbf{P}(t-1) - \frac{\mathbf{P}(t-1)\,\mathbf{x}\,\mathbf{x}^T\,\mathbf{P}(t-1)}{1 + \mathbf{x}^T\,\mathbf{P}(t-1)\,\mathbf{x}} \tag{2.30}$$

So the "Recursive Least Squares" (RLS) algorithm becomes (together with equation (2.29)):

$$\begin{aligned} \mathbf{K}(t) &= \frac{\mathbf{P}(t-1)\,\mathbf{x}(t)}{1+\mathbf{x}^T(t)\,\mathbf{P}(t-1)\,\mathbf{x}(t)} \\ \mathbf{P}(t) &= (\mathbf{I}-\mathbf{K}(t)\,\mathbf{x}^T(t))\,\mathbf{P}(t-1) \end{aligned} \tag{2.31}$$

Typically the initial value of $\mathbf{P}$; $\mathbf{P}(0)$ is taken to be a diagonal matrix $\alpha\mathbf{I}$ where a large value of α (usually 10^4) implies little confidence in $\hat{\boldsymbol{\theta}}(0)$ and gives rapid initial changes in $\hat{\boldsymbol{\theta}}(t)$. If we take α small such as 0.1, it means $\hat{\boldsymbol{\theta}}(0)$ is a reasonable estimate of $\boldsymbol{\theta}$ and as a result $\hat{\boldsymbol{\theta}}(t)$ changes slowly [13].

There is still a question in our minds: $\mathbf{S}(t) = \sum \mathbf{x}(i)\,\mathbf{x}^T(i)$ is such that its norm $\|\mathbf{S}\|$ is initially "small" but tends to ∞, provided that $\mathbf{x}(t)$ is "sufficiently exciting" [13]. Hence $\|\mathbf{K}\|$ goes to 0 and $\hat{\boldsymbol{\theta}}(t) \longrightarrow$ *constant vector* $\boldsymbol{\theta}$. This is acceptable if the "true" parameters were in fact constant, but in practice we would also want the algorithm to track slowly varying parameters (so that the self-tuner stays in tune). So how can we modify our algorithm?

There are many ways to modify RLS to achieve this property, but the most popular is the use of "forgetting factor", β, where $0 < \beta \leq 1$. If we give more weighting to the more recent data instead of giving equal weights to the errors in the LS criterion, the criterion will be [13]:

$$L = \sum_{j=1}^{t} \beta^{t-j}\, e^2(j) \tag{2.32}$$

With this criteria we obtain the following equations for the modified version of RLS:

$$\begin{aligned} \mathbf{K}(t) &= \frac{\mathbf{P}(t-1)\,\mathbf{x}(t)}{\beta+\mathbf{x}^T(t)\,\mathbf{P}(t-1)\,\mathbf{x}(t)} \\ \mathbf{P}(t) &= (\mathbf{I}-\mathbf{K}(t)\,\mathbf{x}^T(t)\,\mathbf{P}(t-1))/\beta \end{aligned} \tag{2.33}$$

In this case $\|\mathbf{P}\|$ and $\|\mathbf{K}\|$ don't tend to zero, so that $\hat{\boldsymbol{\theta}}(t)$ can vary for large t. Although $\hat{\boldsymbol{\theta}}$ is affected by "all" past data , the "asymptotic sample length" (ASL)[8] indicates the number of "important" previous samples contributing to $\hat{\boldsymbol{\theta}}(t)$. In practice, typical values of β are in the range 0.95 (fast variations) to 0.999 (slow variations).

[8] ASL=$\frac{1}{1-\beta}$.

This shows that ASL changes between 20 to 1000 [13].

Now as the introduction to the basic ideas are completed we can continue with the explanation of Minimum Variance control and the self-tuning regulator.

2.3 Minimum Variance Control and the Self Tuning Regulator

In the previous sections we saw that the system model can be written as:

$$C\,y^*(t+k\,h|t) = F\,y(t) + G\,u(t) \tag{2.34}$$

and

$$\begin{aligned} y(t+k\,h) &= y^*(t+k\,h|t) + \tilde{y}(t+k\,h|t) \\ &= y^*(t+k\,h|t) + E\,\xi(t+k\,h) \end{aligned} \tag{2.35}$$

Now, consider a control law which chooses $u(t)$ to minimize the variance $I = E\{y^2(t+k\,h)\}$, where the expectation is conditioned on all data over the range $(-\infty, t)$ and the ensemble is all realizations of the random processes $\xi(t+i\,h)$ affecting the output after time t [13]. Now one can write:

$$\begin{aligned} I = E\{y^2(t+k\,h)\} &= E\{(y^*(t+k\,h|t) + \tilde{y}(t+k\,h|t))^2\} \\ &= (y^*(t+k\,h|t))^2 + E\{(\tilde{y}(t+k\,h|t))^2\} \end{aligned} \tag{2.36}$$

because y^* and $\tilde{y}$ are orthogonal (see [13]) and y^* is known at time t.

As $\tilde{y}$ is unaffected by $u(t)$, the minimum I is clearly when y^* is set to zero by the choice of control:

$$F\,y(t) + G\,u(t) = 0 \tag{2.37}$$

or

$$u(t) = -\frac{F\,y(t)}{B\,E} \tag{2.38}$$

Note that the closed loop (Figure 2-1) characteristic equation is:

$$1 + \frac{F}{BE}\frac{z^{-k}B}{A} = 0 \tag{2.39}$$

or

$$B(EA + z^{-k}F) = 0 \tag{2.40}$$

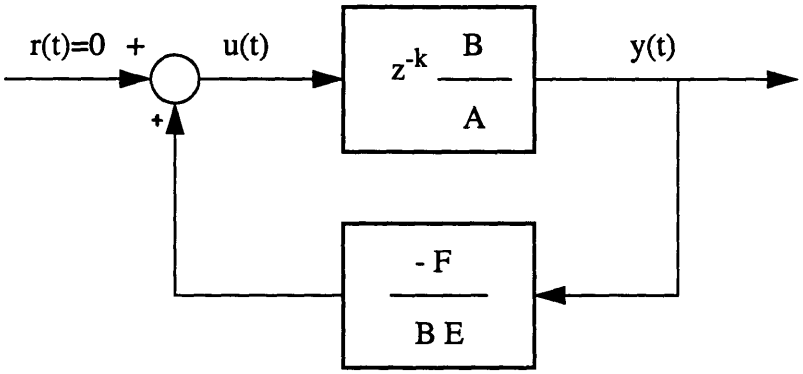

Figure 2-2: Simple closed loop sketch of minimum variance control

By using identity we obtain:

$$B(z^{-1})C(z^{-1}) = 0 \tag{2.41}$$

As C is a stable polynomial, it's seen that the closed loop stability depends on the sampled-process zeros. However, there are many cases (eg. with fast sampling or large fractional delay) where B has roots outside the stability region [13]. In these applications, minimum variance control should be used with caution. (Full block diagram of minimum variance self-tuning control can be seen in Figure 2-3.)

The original self-tuning regulator of Åström and Wittenmark (1978) used the minimum variance objective function [13]. Consider, first, equations (2.15) and (2.16) for the case where $C(z^{-1}) \equiv 1$, and time t rather than $t + k$ (for simplicity assume $h = 1$ sec.):

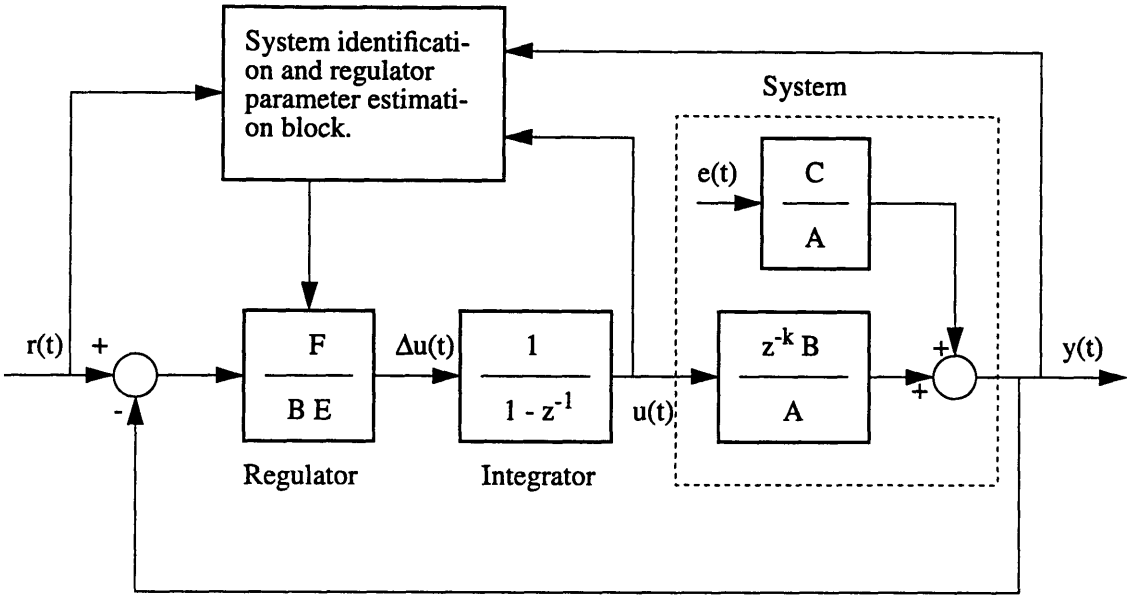

Figure 2-3: Block diagram representation of Minimum Variance Self-tuning control.

$$y(t) = F\,y(t - k\,h) + G\,u(t - k\,h) + E\,\xi(t) \tag{2.42}$$

So;

$$\left.\begin{aligned} \boldsymbol{\theta}^T &= (f_0, f_1, \ldots, f_{n-1}, g_0, g_1, \ldots, g_{n+k-1}) \\ \mathbf{x}^T(t) &= (y(t-k), \ldots, u(t-k), \ldots, u(t-k-n+1)) \\ \varepsilon(t) &= \xi(t) + e_1\,\xi(t-1) + \ldots + e_{k-1}\,\xi(t-k+1) \\ \phi(t) &= y(t) \end{aligned}\right\} \tag{2.43}$$

Here, $\varepsilon(t)$ is autocorrelated process, but is independent of "known data", $\mathbf{x}(t)$. Hence RLS can be used to obtain unbiased (though not optimal) estimates of $\hat{\boldsymbol{\theta}}$ [13]. Then the certainty-equivalent control law becomes:

$$\hat{F}\,y(t) + \hat{G}\,u(t) = 0 \tag{2.44}$$

This is an "implicit" self tuner, as the required feedback parameters are estimated directly rather than via a control-design calculation as we discussed in the first chap-

ter. We know that the parameter estimates in equation (2.42) are not unique[9]. But, we can fix this problem by several methods that are discussed in the literature [13], or we can also just ignore the lack of uniqueness in the estimation. Although this complicates the convergence analysis, according to Clarke, the self-tuning performance is still proved to be effective in most applications [13].

Now consider the case when $C(z^{-1}) \neq 0$ but a general polynomial in the form:

$$\frac{1}{C(z^{-1})} = 1 + \gamma_1 z^{-1} + \gamma_2 z^{-2} + \cdots = C^{-1} \tag{2.45}$$

This time, the predictor model can be rewritten as [13]:

$$y(t) = F\,y(t-k) + G\,u(t-k) + \gamma_1 \left(F\,y(t-k-1) + G\,u(t-k-1)\right) + \cdots + E\,\xi(t) \tag{2.46}$$

If $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$ then the control $(\hat{F}\,y(t) + \hat{G}\,u(t) = 0)$ sets all the terms on the right hand side to 0 so that the equation reduces to:

$$y(t) = F\,y(t-k) + G\,u(t-k) + E\,\xi(t) \tag{2.47}$$

as if $\gamma_i = 0$ and hence if $C \equiv 1$. This implies that $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$ is a fixed point of the algorithm [13]. In the initial tuning stage, however $\hat{\boldsymbol{\theta}}$ may be remote from $\boldsymbol{\theta}$ and the γ_i's may have a significant effect on the "rate" of convergence. Clarke states that, intuitively, the dynamics of $1/C$ and the convergence rate are related [13]. As the stability and convergence properties of control systems is important, in the next section, I would like to present the basic and important points of the stability and convergence properties of the Minimum Variance self-tuning control.

[9]Because we can multiply this equation by any arbitrary constant, without affecting the calculation of $u(t)$.

2.4 Stability and Convergence of Self-tuners

Stability (in the sense of giving rise to bounded control signal and system output), and convergence(in the sense that the desired system performance is asymptotically achieved) are desirable properties of any adaptive controller. That these properties, under certain conditions, apply to the implicit algorithms is an important feature of the self-tuning method. Now, lets discuss the stability and the convergence of Minimum Variance self-tuning control.

2.4.1 Overall Stability

From the point of view of applications, stability is the most important property. Without the stability the regulators would be useless. The stability of a stochastic system can be defined and analyzed by many different ways. But the easiest way is, according to Åström, perhaps to make a linearization and to consider small perturbations from an equilibrium point of the system [4]. But from the practical point of view this is not of a great value because the system may still depart from the region where the linearization is valid. So a global stability concept which guarantees that the solutions remain bounded with probability one is much more useful. The only drawback of this solution can be, however, the bounds obtained may be larger than can be accepted .

The stability analysis of minimum variance controller with least squares parameter estimation was considered in many researches in the literature[10] [4, 24]. In these researches it is shown that:

$$\lim_{N\to\infty} \sup \frac{1}{N} \sum_{i=1}^{N} [y(t)^2 + u(t)^2] < \infty \tag{2.48}$$

with probability one, if the time delay k of the process is known and if the order of

[10]For example the special case of a regulator based on least squares estimation and minimum variance control applied to minimum phase processes is analyzed in; L. Ljung and B. Wittenmark. On a stabilizing property of adaptive regulators. IFAC Symposium on Identification and System Parameter Estimation, Tbilisi, 1976.

the system is not underestimated. The only requirement on the disturbance($d(t)$) is that

$$\lim_{N\to\infty} \sup \frac{1}{N} \sum_{i=1}^{N} d(t)^2 < \infty \tag{2.49}$$

with probability one, which is a fairly weak condition and does not necessarily involve stochastic framework [4].

The result is important because it states that the particular regulator (Least Squares together with Minimum Variance) will stabilize any linear time invariant (LTI) process provided the conditions given above are satisfied.

2.4.2 Convergence

The asymptotic behavior of the regulators can be studied by using the results for the convergence analysis of general recursive, stochastic algorithms[11]. Convergence analysis of self-tuning control algorithms can also be found in [37, 24].

There can be two possibilities for convergence;

a) *Possible convergence points when the structure of the model is compatible with that of the system.*

Control engineers say, model is compatible with the system if the disturbance is a moving average of an order corresponding to the assumed order of the C-polynomial. In particular, if the least squares scheme is used ($C(z) \equiv 1$) then disturbance has to be white-noise [4].

To study if the true parameter values $\boldsymbol{\theta}_0$ is a possible convergence point is of particular interest, since they give the optimal regulator. In most of the identification methods (LS, RLS, RML, etc.) we can show that the estimated parameters converge to the real values. But there are cases when extended least squares (ELS) is used, where we cannot have convergence to the desired limit

[11] L. Ljung. Convergence of recursive stochastic algorithms. IFAC Symposium on Stochastic Control, Budapest, 1974. And,
L. Ljung. Analysis of recursive stochastic algorithms. *IEEE Transactions on Automatic Control.* AC-22, 1977.

($\equiv$ the true values)[12] [4].

b) *Possible convergence points when the model is not compatible with the system*[13]. In this general case, it is usually pretty difficult to say anything. As we do not have any true parameter values, it cannot be expected that we would have convergence to parameters which yield the best regulator. However, in the special case of least squares identification combined with minimum variance control there is quite a remarkable result [5, 4]: If in the true system[14] disturbance is a moving average of order n and the model[15] orders are chosen as $p = n + k$, $r = n + 2k - 1$ and $s = 0$ then there is only one point that gives minimum variance control. Therefore, if this regulator converges, it must converge to the minimum variance regulator. More detailed information and proofs can be found in either [4] or [24].

2.5 Explicit and Implicit Self-tuning Algorithms

In the previous sections I used the phonemenon "implicit algorithm" for the Minimum Variance Control. In this section, I would like to explain what these implicit and explicit terms mean.

2.5.1 Explicit or Indirect Self-tuning Algorithm

When using an explicit algorithm (Figure 2-3), an explicit process model is estimated. (i.e. the coefficients of the polynomials $A^\star$, $B^\star$ and $C^\star$ in a system described as $A^\star\, y(k) = B^\star\, u(k-d) + C^\star\, e(k)$).

An explicit algorithm can then be described in two steps. The first step is to estimate the polynomials $A^\star$, $B^\star$ and $C^\star$ of the process model that is described by

[12] For this discussion please refer to:
L. Ljung, T. Söderstöm and I. Gustavsson. Counter examples to general convergence of a commonly used identification method. *IEEE Transactions on Automatic Control*, AC-20:643–652, 1975.

[13] This discussion is partly taken from [4].

[14] $A(z^{-1})\, y(t) = B(z^{-1})\, u(t-k-1) + d(t)$ where h is taken as 1 sec.

[15] $\hat{y}(t) = -\mathcal{A}(z^{-1})\, y(t-1) + \mathcal{B}(z^{-1})\, u(t-1) + \mathcal{C}(z^{-1})\, \xi(t-1)$, where the polynomial $\mathcal{A}, \mathcal{B}, \mathcal{C}$ ends with z^{-p+1}, z^{-r+1} and z^{-s+1} respectively. For simplicity h is chosen to be 1 sec.

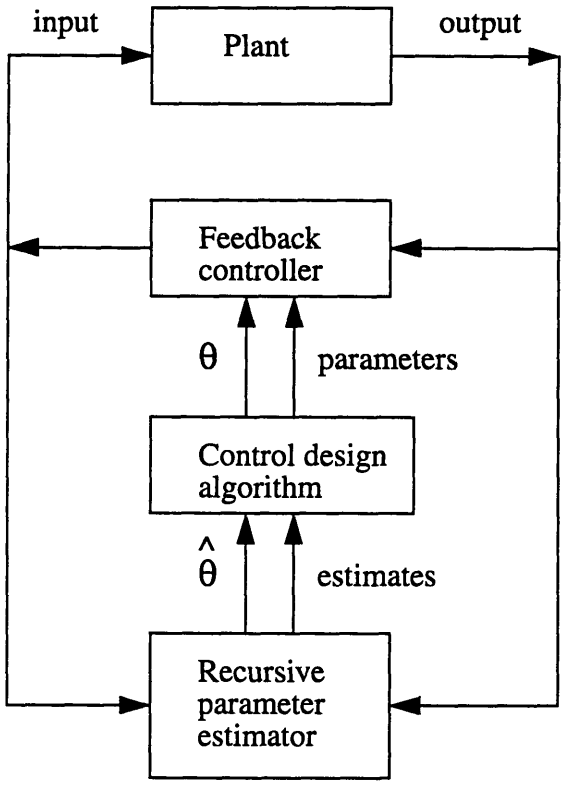

Figure 2-4: Structure of an explicit self-tuner

the above equation. In the second step, a design method is used to determine the polynomials in the regulator (such as $R^{\star}\,u((k) = -S^{\star}\,y(k) + T^{\star}\,u_c(k)$) using the estimated parameters from the first step, where u_c is the reference value or the command signal. The two steps are repeated at each sampling interval. The design procedure in the second step can be any good design method that is suitable for the problem that we are dealing with. One of the commonly used design procedure is the "Pole Placement Algorithm", which I want to go over very briefly because of its special form.

2.5.2 Explicit Pole-placement Algorithms

In the previous sections we saw that, in order to achieve its performance, the minimum variance controller cancels the system dynamics and is highly sensitive to the positions

of system zeros [13]. (To reduce this sensitivity we have several methods [13]). And the system delay k must be known, so that a k step ahead predictor model can be formed and used.

In an explicit algorithm, on the other hand, the standard system model is estimated; moreover a range of time delays can be accommodated by overparameterizing the $B(z^{-1})$ polynomial at the cost of extra computations [13].

If k is known to be in the range $[k_1, k_2]$; then if $\beta(z^{-1})$ is a polynomial with $n + k_2 - k_1$ terms, the identified model is;

$$A\,y(t) = \beta\,u(t - k_1\,h) + C\,\xi(t) \tag{2.50}$$

If k were in fact k_1, the first coefficients of $\hat{\beta}$ would be non-zero, whereas if k were k_2 the last n coefficients would be non-zero. Hence $\hat{\beta}$ can be used in a self-tuning algorithm provided that the control design is insensitive to the resultant gross variations in the zeros of $\hat{\beta}$; this insensitivity can be at the cost of ultimate performance[16].

2.5.3 Implicit or Direct Self-tuning Algorithms

In an implicit algorithm (Figure 2-4), the parameters of the regulator are estimated directly. This can be made possible by a reparameterization of the process model. A typical example is the minimum variance self-tuner.

One advantage with the implicit algorithms over the explicit ones is that the design computations are eliminated, since the controller parameters are estimated directly. The implicit algorithms usually have more parameters to estimate than the explicit algorithms, especially if there are long time delays in the process.

Simulations and practical experiments indicate, however, that the implicit algorithms are more robust [6]. On the other hand, the implicit algorithms usually have the disadvantage that all process zeros are canceled. According to Åström, this implies that the implicit methods are intended only for processes with a stable inverse or minimum phase systems. Sampling of a continuous time systems often gives a discrete

[16]For further and detailed discussion, the reader should refer to [13, 31].

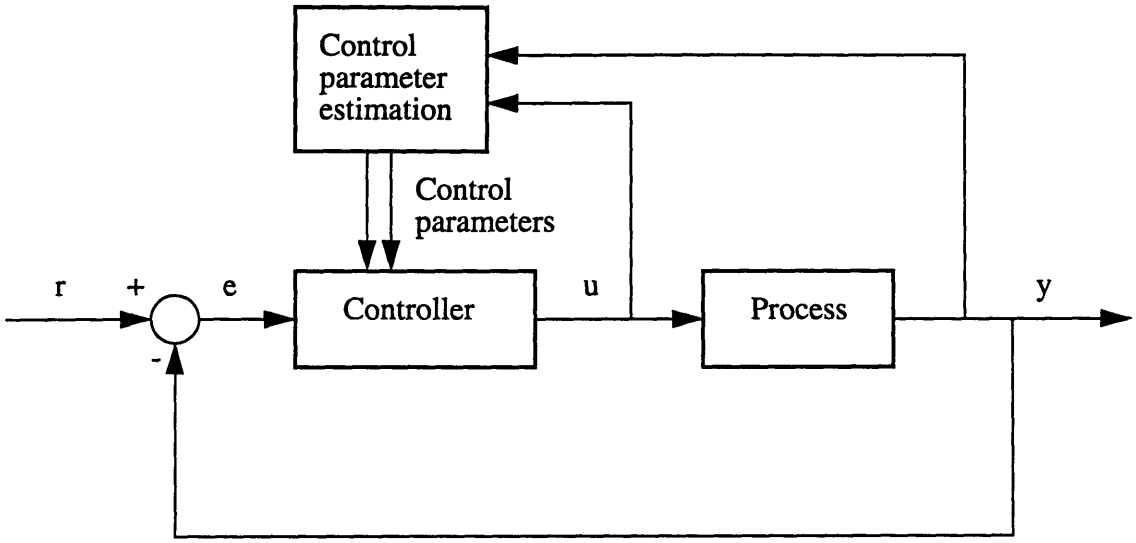

Figure 2-5: Structure of an implicit self-tuner

time system with zeros on the negative real axis, inside or outside the unit circle. It's not good to cancel these zeros even if they are inside the unit circle, because cancellation of these zeros will give rise to "ringing" in the control signal [64]. Many implicit algorithms can, however, be used also if the system is nonminimum phase through a proper choice of parameters. For further discussion, refer to [4, 5, 15, 6, 63].

2.6 Applications of Self-tuning control

Self-tuning control theory can be used in many different ways. In previous sections we observed that the regulator becomes an ordinary constant gain feedback. So if the parameter estimates are kept constant, the self-tuner can be used as a special tuner to adjust the control loop parameters. In this kind of applications, the self-tuner runs until satisfactory performance is obtained. Then it is turned off and the system is left with the constant parameter regulator.

Self-tuning control can also be used to obtain or build up a gain schedule. In this case, the regulator is run at different and wide range of operating points and the controller parameters obtained are stored in memory. Then a table of parameters can be constructed and used for the process for any arbitrary operating point by the use

of interpolation of the values previously obtained.

Another application of self-tuners can be as a real adaptive controller for systems with varying parameters. According to Narenda, if operating conditions are widely varying, combinations of gain-scheduling and self-tuning can also be applied [2].

2.7 Misuses of Self-tuning Control

Until this point all the positive points of self-tuning control were described. But as any other controller, a self-tuner can be used in an inappropriate manner.

First of all, the word self-tuning may lead to the false conclusion that these controllers can be switched on and used without any a priori considerations. But this is definitely not true. If we compare the self-tuner with a three term controller (PID), it can easily be seen that it is a sophisticated controller. So before it can be switched on, many important points such as underlying design and estimation methods, initialization, selection of parameters, etc. should be considered.

Secondly, as it is a fairly complex control law, it should not be used if a simpler PID controller can do the job. Before deciding on self-tuning control, therefore, it is useful to check whether a constant parameter regulator or any other simple controller is sufficient.

As a last point, it is always good to go through the particular application carefully and decide upon a design method which is suitable for the particular problem. Also the model for the process and the environment should be considered thoroughly in order to reduce possible faults [2].

In this chapter the basic formulation of Minimum Variance self-tuning control together with recursive parameter estimation is given. Having this background, the simulations that were prepared to show the results of Minimum Variance Control will be introduced in the next chapter.

Chapter 3

Programming and Simulations

In the second chapter, the basic formulation for system identification (**R**ecursive **L**east **S**quares) and Self-tuning Control (Minimum Variance Control) was given. With the help of these basics, simple programs using Matlab^{TM} were prepared. Then, by using the excellent features of Matlab^{TM} on matrix manipulation, these programs were generalized as much as possible. This chapter, consists of two parts, in the first part the programs on self-tuning control and how they work, together with sample simulations will be introduced. In the second part, the modelling scripts that were prepared for this thesis and for self-tuning control applications will be given.

3.1 A Simple Program

Appendix A.1 describes a program on parameter estimation and simulation of a well-known system. The model in this program is in the form:

$$\frac{Y(s)}{U(s)} = \frac{e^{-\Delta s}(\tau_1 s + 1)}{(\tau_2 s + 1)(\tau_3 s + 1)} \tag{3.1}$$

This script, first converts the system to discrete time (either by Tustin or usual ZOH, depending on the choice), since all the calculations that are used for self-

tuning is in discrete time framework[1]. At this point it makes a simulation and gets the necessary input and output data for Recursive Least Squares. Then the RLS algorithm runs, and finds the parameter estimates as seen in Figure 3-1. The discrete time representation of this system with parameters can be described as:

$$G(z) = \frac{b_0 z^2 + b_1 z + b_2}{z^4 + a_1 z^3 + a_2 z^2} \tag{3.2}$$

or

$$G(z^{-1}) = z^{-2} \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{3.3}$$

After that, the program reconstructs the system from the parameter estimates and pursues another simulation involving the true and estimated parameters. The result of this simulation can be seen in Figure 3-2.

This script was rerun with different types of transfer functions, and in each case the results seemed to be satisfactory, in the sense that the model and the reconstructed system behaved similarly. These simulations confirmed that the RLS algorithm is implemented correctly, so by using the same logic,the General Parameter Estimation script (Appendix A.2), that finds the parameter estimates of any system using Recursive Least Squares algorithm with exponential weighting factor was prepared. For the users that prefer plain Least Squares algorithm, a short script that can be directly applied to systems is given in Appendix A.2.1.

3.2 General Minimum Variance Self-tuning Control

After having read several implementations [4, 54, 13, 43, 14, 62, 44, 23, 10, 21, 1] and algorithms [30, 41, 37, 17, 11, 42] on Minimum Variance Control, a general self-tuning program together with parameter estimation and white noise aspects was prepared.

[1]This program also creates a random input (see Appendix A.1.1) that will be used in the simulations.

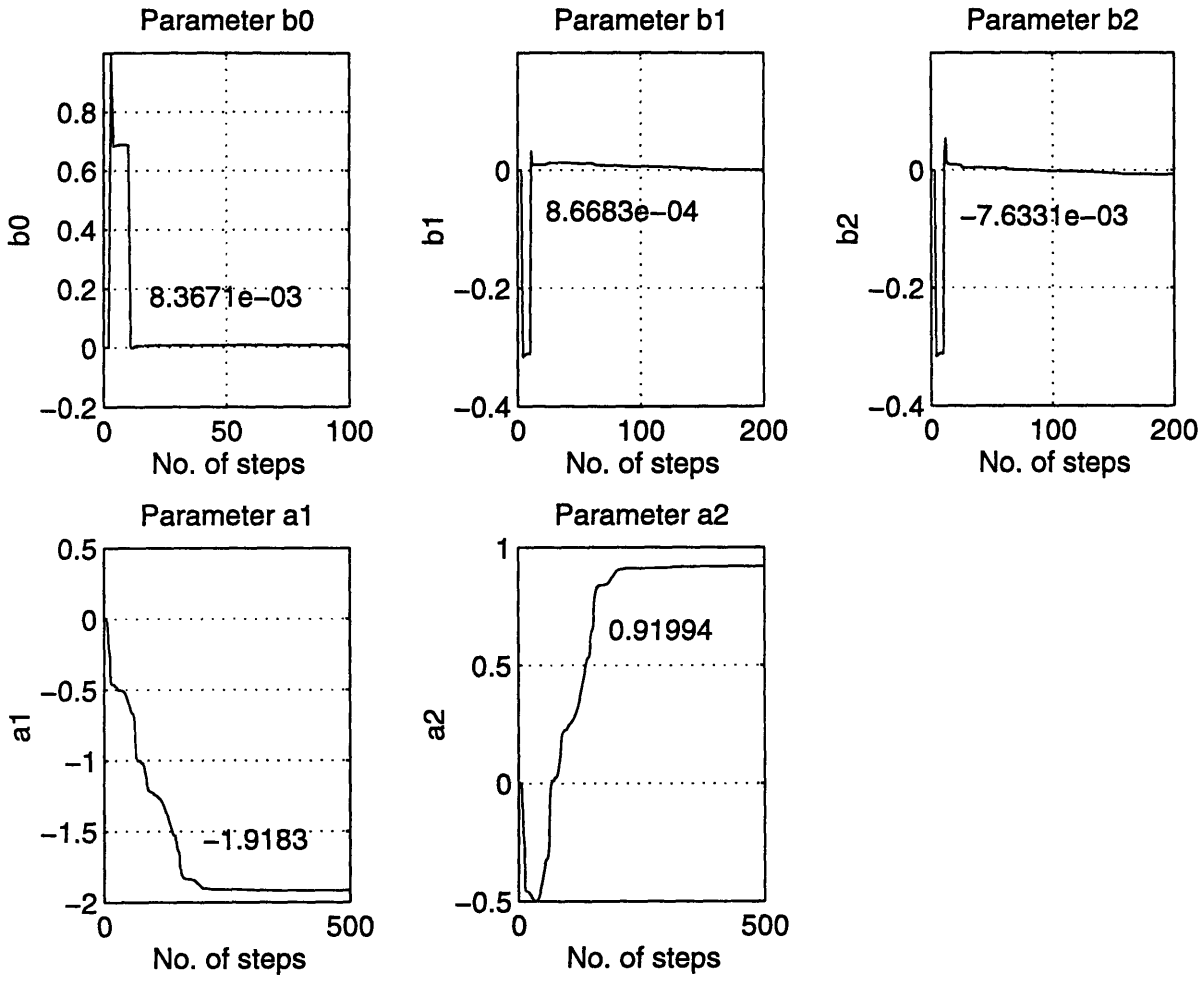

Figure 3-1: Typical parameter estimation output of RLS algorithm

However, the formulation of the minimum variance control algorithm that was used here is a little bit different from the one that was described in Chapter 2. Because of this reason, after giving an example for the basic method, the one that was used in these programs will be reviewed.

3.2.1 Minimum Variance Control

In what follows, a simple example is given to illustrate the minimum variance control algorithm.

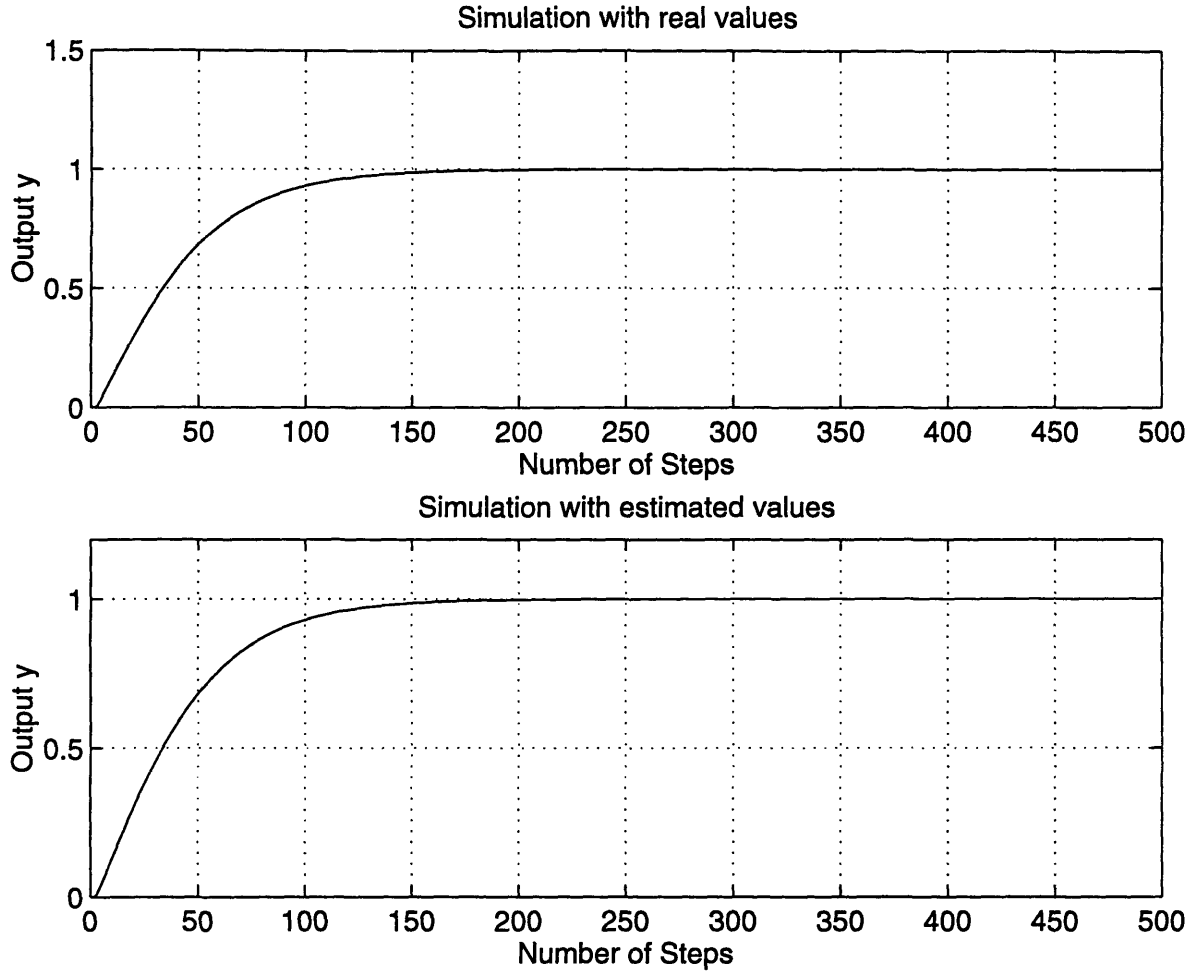

Figure 3-2: Comparison of real system output and model output

Let us consider the system described in [13] with $h = 1$ sec.:

$$(1 - 0.9z^{-1})\, y(t) = 0.5\, u(t-2) + (1 + 0.7z^{-1})\, \xi(t) \tag{3.4}$$

where

$$A(z^{-1}) = 1 - 0.9z^{-1} \tag{3.5}$$

$$B(z^{-1}) = 0.5 \tag{3.6}$$

and

$$C(z^{-1}) = 1 + 0.7z^{-1} \tag{3.7}$$

From these equations, the aim is to find the minimum variance control law. But

first, let's review the general solution:

The general description of a discrete time system, as we saw in the previous chapter, can be written as:

$$y(t+k\,h) = \frac{B(z^{-1})}{A(z^{-1})}\,u(t) + \frac{C(z^{-1})}{A(z^{-1})}\,\xi(t+k\,h) \tag{3.8}$$

or

$$A(z^{-1})\,y(t+k\,h) = B(z^{-1})\,u(t) + C(z^{-1})\,\xi(t+k\,h) \tag{3.9}$$

If we multiply both sides of this equation by $E(z^{-1})$ which has the degree of $k-1$ (with $E(0)=1$), and form the *Diaphontine* equation as $C(z^{-1}) = E(z^{-1})\,A(z^{-1}) + z^{-k}\,F(z^{-1})$ with degree of $F(z^{-1}) = n-1$, one obtains:

$$E(z^{-1})\,A(z^{-1})\,y(t+k\,h) = E(z^{-1})\,B(z^{-1})\,u(t) + E(z^{-1})\,C(z^{-1})\,\xi(t+k\,h) \tag{3.10}$$

$$C(z^{-1})\,y(t+k\,h) - F(z^{-1})\,y(t) = G(z^{-1})\,u(t) + E(z^{-1})\,C(z^{-1})\,\xi(t+k\,h) \tag{3.11}$$

where $G(z^{-1}) = E(z^{-1})\,B(z^{-1})$. We can rewrite this equation as:

$$y(t+k\,h) = \frac{F(z^{-1})\,y(t) + G(z^{-1})\,u(t)}{C(z^{-1})} + E(z^{-1})\,\xi(t+k\,h) \tag{3.12}$$

Now, let $y^*(t+k\,h|t)$ be the optimum prediction where $error = \tilde{y}(t+k\,h|t)$ and $y = y^* + \tilde{y}$. Then, it was seen before that one can write:

$$C(z^{-1})\,y^*(t+k\,h|t) = F(z^{-1})\,y(t) + G(z^{-1})\,u(t) \tag{3.13}$$

and

$$\tilde{y}(t+k\,h|t) = E(z^{-1})\,\xi(t+k\,h) \tag{3.14}$$

So the minimum variance control law, as described in chapter two (equation (2.35) and (2.36)) becomes:

$$u(t) = -\frac{F(z^{-1})\,y(t)}{B(z^{-1})\,E(z^{-1})} \tag{3.15}$$

In the above example, $k = 2$ and $n = 1$. So Diaphontine equation will be:

$$1 + 0.7z^{-1} = (1 + e_1\, z^{-1})(1 - 0.9z^{-1}) + z^{-2}\, f_0 \tag{3.16}$$

After solving this equation, one obtains $e_1 = 1.6$ and $f_0 = 1.44$. And the minimum variance control law is:

$$u(t) = -\frac{1.44\, y(t)}{0.5(1 + 1.6z^{-1})} \tag{3.17}$$

This result can also be written as:

$$u(t) = -1.6\, u(t-1) - 2.88\, y(t)$$

3.2.2 General Description For Implicit Self-tuner

Now let's form the self-tuning algorithm that is used in this thesis[2]. For simplicity in understanding, the sample interval (h) is chosen to be 1 sec.

In general a system can be described by the following equations:

$$A(z^{-1})\, y(t) = B(z^{-1})\, u(t-d) + C(z^{-1})\, \xi(t) \tag{3.18}$$

or

$$A(z^{-1})\, y(t) = z^{-d}\, B(z^{-1})\, u(t) + C(z^{-1})\, \xi(t) \tag{3.19}$$

where,

$$\left.\begin{array}{rcl} A(z^{-1}) & = & 1 + \sum_{j=1}^{n} a_j z^{-j} = 1 + a_1 z^{-1} + \cdots + a_n z^{-n} \\ B(z^{-1}) & = & b_0 + \sum_{j=1}^{m} b_j z^{-j} = b_0 + b_1 z^{-1} + \cdots + b_m z^{-m} \\ C(z^{-1}) & = & 1 + \sum_{j=1}^{k} c_j z^{-j} = 1 + c_1 z^{-1} + \cdots + c_k z^{-k} \end{array}\right\} \tag{3.20}$$

So if we write the difference equation:

$$y(t) = b_0\, u(t-d) + \cdots + b_m\, u(t-d-m) - a_1\, y(t-1) - \cdots - a_n\, y(t-n) + \xi(t) \tag{3.21}$$

[2]The method of solving Diaphontine equation could also be used, but the following representation is much simpler for programming purposes. For implementing Diaphontine equation solution technique please refer to [14]. (In particular, chapter four, part 3 of this collection, pages 77–83).

We can calculate:

$$y(t) = \boldsymbol{\theta}(t)\,\boldsymbol{\varphi}(t-1) + \xi(t) \tag{3.22}$$

where

$$\boldsymbol{\theta} = [b_0 \;\; b_1 \;\; \cdots \;\; b_m \;\; -a_1 \;\; -a_2 \;\; \cdots \;\; -a_n] \tag{3.23}$$

$$\boldsymbol{\varphi}(t-1) = [u(t-d) \;\; u(t-d-1) \;\; \cdots \;\; u(t-d-m) \;\; y(t-1) \;\; y(t-2) \;\; \cdots \;\; y(t-n)]^T \tag{3.24}$$

So the expected output becomes:

$$\hat{y} = \hat{\boldsymbol{\theta}}(t)\,\boldsymbol{\varphi}(t-1) \;\;,\;\; t = 1, \cdots, n \tag{3.25}$$

where (for the RLS algorithm):

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1) + (y(t) - \overbrace{\hat{\boldsymbol{\theta}}(t-1)\,\boldsymbol{\varphi}(t-1)}^{\hat{y}})\,\boldsymbol{\varphi}^T(t-1)\,\mathbf{R}(t)) \tag{3.26}$$

$$\mathbf{R}(t) = \mathbf{R}(t-1) - \frac{\mathbf{R}(t-1)\,\boldsymbol{\varphi}(t-1)\,\boldsymbol{\varphi}^T(t-1)\,\mathbf{R}(t-1)}{1 + \boldsymbol{\varphi}^T(t-1)\,\mathbf{R}(t-1)\,\boldsymbol{\varphi}(t-1)} \tag{3.27}$$

The initial value ($\mathbf{R}(t-1)$) is usually taken as $10^4\,\mathbf{I}$ or $10^5\,\mathbf{I}$ in the beginning of simulation. In self-tuning control the quadratic cost function can also look like [14] (notice the difference from the second chapter):

$$J(t) = E[y(t+d|t) - y_r]^2\,Q_1 + [u(t)]^2\,Q_2 \tag{3.28}$$

Here weight is put on both the error and the control input to make the algorithm more flexible. To calculate our optimum control signal the following calculation can be done:

$$\left.\begin{aligned} E[y(t+d|t)] &= \hat{y}(t+d|t) = \hat{\boldsymbol{\theta}}(t)\,\boldsymbol{\varphi}(t+d-1) \\ &= \hat{b_0}\,u(t) + \hat{b_1}\,u(t-1) + \cdots + \hat{b_m}\,u(t-m) - \cdots \\ &\quad -\hat{a_1}\,\hat{y}(t-1+d)|_t \cdots - \hat{a}_{d-1}\,y(t-d+1) - \cdots \\ &\quad \cdots - \hat{a}_0\,y(t) - \cdots - \hat{a}_n\,y(t-n+d) \end{aligned}\right\} \tag{3.29}$$

So:

$$\hat{y}(t+d|t) - y_r = \hat{b_0}\, u(t) + \hat{c}(t) \tag{3.30}$$

where

$$\hat{c}(t) = \sum_{j=1}^{m} \hat{b}_j\, u(t-j) - \sum_{j=1}^{d-1} \hat{a}_j\, \hat{y}(t-j+d|t) - \sum_{k=d}^{n} \hat{a}_k\, y(t-k+d) - y_r \tag{3.31}$$

So, if we form our cost function again using these results, we get:

$$J(t) = [\hat{b}_0\, u(t) + \hat{c}(t)]^2\, Q_1 + [u(t)]^2\, Q_2 \tag{3.32}$$

and optimization for $u(t)$ gives:

$$\frac{\partial J}{\partial u} = 0 \quad \Rightarrow \quad u(t) = -\frac{\hat{c}(t)\, Q_1\, \hat{b}_0}{\hat{b}_0^2\, Q_1 + Q_2} \tag{3.33}$$

or:

$$u^o(t) = -(\hat{b}_0^2\, Q_1 + Q_2)^{-1}\, \hat{b}_0\, \hat{c}(t)\, Q_1 \tag{3.34}$$

Using the above algorithm, the general self-tuning control program (Appendix A.3) was prepared, which generates the necessary input command $u(t)$ in order to reach the desired reference values. In order to prevent any steady state errors an integral action was also added in the control law which is commonly used in multi input multi output(MIMO) algorithms[3].

After successfully completing the script, several demonstrations and simulations including a real system was constructed. In the following sections, I would like to introduce these demonstrations and their results.

[3]For this addition the reader should refer to Appendix 3.1.

3.3 Simulations

3.3.1 An SISO System

The first simulation that was prepared is the same SISO system that was presented in the previous section (equation (3.1)). In this case, the weighting factors $Q_1 = 1$, and $Q_2 = 0.001$ was chosen. The maximum and the minimum values for the input command was chosen to be -10 and 10 respectively. Other variables and initial conditions are selected as follows: Weighting factor(β)=0.98, time interval(h)=0.01 sec., standard deviation for the noise is assumed to be 0.001 and mean is 0, delay(Δ)=0.015 sec., initial value(y_0)=0, and the reference trajectory (y_r) was chosen to be 1 for the first hundred steps and -1 for the next hundred steps. For the identification part n=3, m=2 and k=0 were assumed. After preparing the necessary script (Appendix A.4.2) for the general program, the simulation was run.

The output of this simulation can be seen in Figure 3-3. If we examine the output we see that the parameters that was chosen for the system were appropriate, and the system approaches the desired values (1 and -1). The command input signal is also very reasonable. It reaches the saturation in two parts (roughly from 6th to 10th steps and from 102 to 110th steps) but it can continue to command the system without diverging.

This first simulation was also performed with different references and parameters and each time a good response was obtained. The system was also forced to its limits by giving a high delay, a bigger forgetting factor, and a smaller saturation values for the input, and the simulation program still worked well except for very high delays, and very low saturation values (such as 1 and -1).

3.3.2 An MIMO System In Time Series Form

After having successfully completed a SISO continuous time system, a MIMO (in time series form) system was constructed for the algorithm. The system used is the model of a cement process. This system has two inputs and two outputs (Appendix

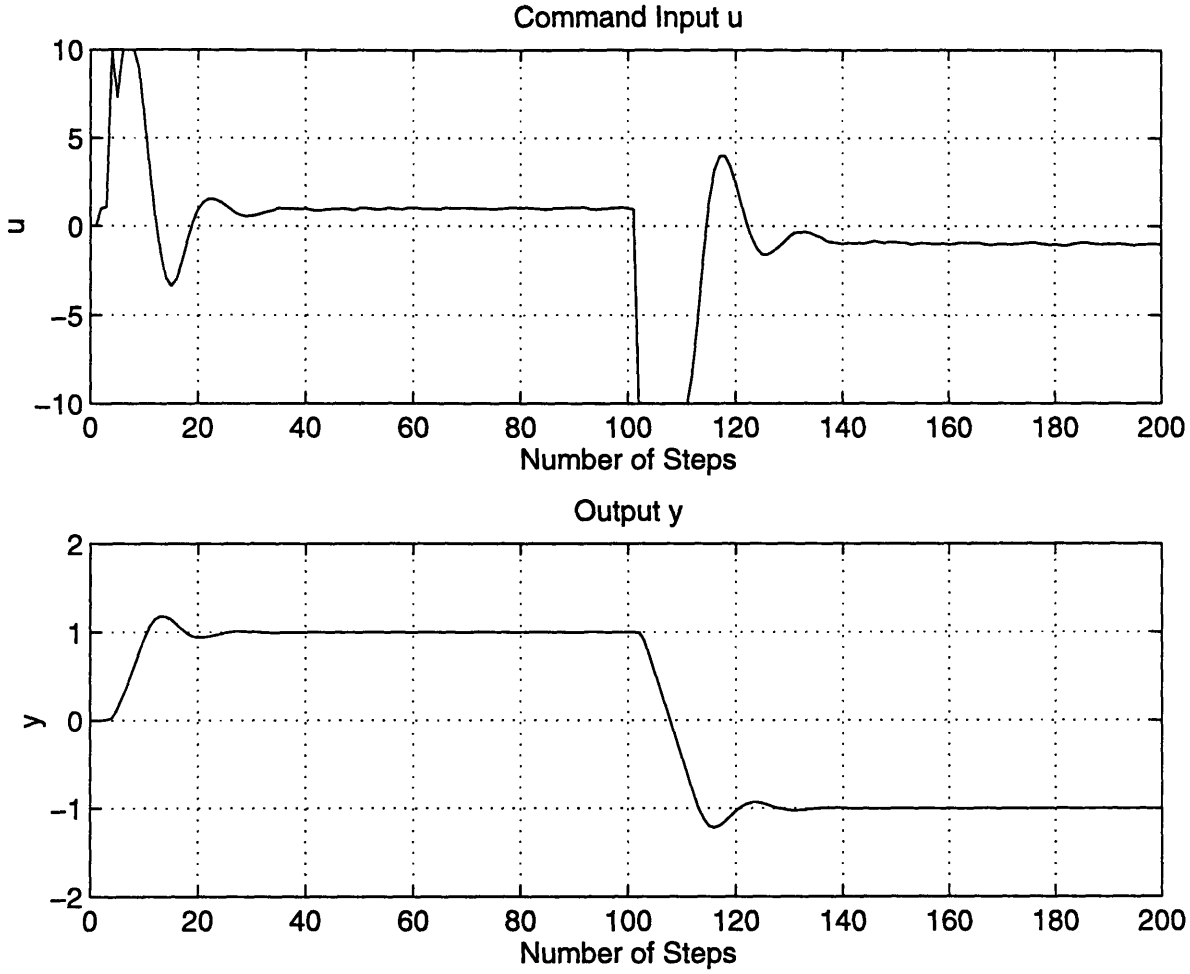

Figure 3-3: Self-tuning simulation output of a well-known system

A.4.3) and has the form:

$$(\mathbf{I} + \mathbf{A}_1 z^{-1} + \mathbf{A}_2 z^{-2})\,\mathbf{y}(z^{-1}) = (\mathbf{B}_0 + \mathbf{B}_1 z^{-1})\,\mathbf{u}(z^{-1}) + \mathbf{e}(z^{-1}) \tag{3.35}$$

where

$$\mathbf{A}_1 = \begin{bmatrix} -1.5 & 0.3 \\ 0.2 & -1.5 \end{bmatrix} \tag{3.36}$$

$$\mathbf{A}_2 = \begin{bmatrix} 0.54 & -0.1 \\ 0.1 & 0.56 \end{bmatrix} \tag{3.37}$$

$$\mathbf{B}_0 = \begin{bmatrix} 2 & -0.3 \\ 0.1 & 0.56 \end{bmatrix} \tag{3.38}$$

and

$$\mathbf{B}_1 = \begin{bmatrix} -1.8 & 0.2 \\ -0.2 & -0.5 \end{bmatrix} \tag{3.39}$$

For this system the necessary file was written and then the simulation was run. The initial values were assumed to be: β=0.98, Q_1=1, Q_2 = 0.001, maximum and minimum values for the command input u was taken to be 50 and -50 respectively, a delay of 1 step is assumed, initial value of the output ($\mathbf{y}_0$) is taken as 0, and the reference trajectory was selected to be 1 for each output. The result of this simulation can be seen in Figure 3-4.

The simulation results indicate that the general program is working well in both continuous and time series mode. Now, the application of this program to a real system will be introduced.

3.3.3 Application to a real system

The system that was chosen here is the GE T700 Turboshaft Helicopter engine[4].

A conventional helicopter utilizes a simple main rotor, primarily for lift, and a tail rotor for torque reaction directional control in the yaw degree of freedom. The main and the tail rotor systems are directly coupled to two turboshaft engines through gear reduction sets and shafting. More information about the system and the model can be found in [47].

In this study, the reduced order model of the engine was used [7] to control only one parameter, power turbine speed (NP). The file, together with initial conditions, system equations and all other variables can be checked in Appendix A.4.4., and the

[4]The following application of this system to self-tuning control was accomplished in the Mechanical Engineering Department of Istanbul Technical University under the supervision of Professor Can Özsoy.

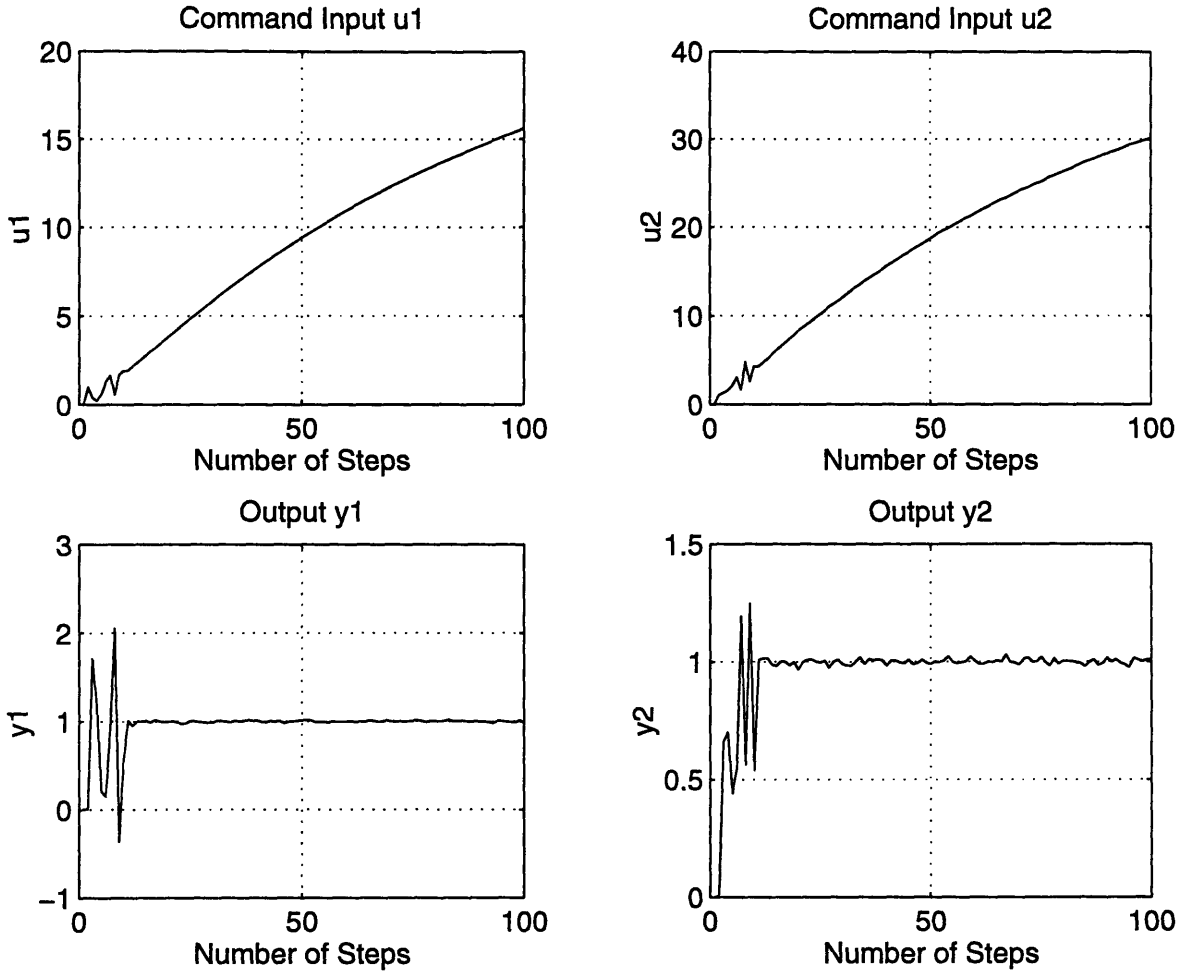

Figure 3-4: Self-tuning simulation output of a two input-two output system

results of the self-tuning simulation can be seen through Figures 3-5 to 3-7. Figure 3-5 shows the command input that was computed from the self-tuning algorithm, the disturbance trajectory, behavior of the power turbine speed (controlled parameter) together with the reference trajectory and the behavior of the gas turbine speed.

The results of this simulation are acceptable. The performance of the turbine speed (with disturbance) remains reasonable, as we wanted, and the system follows the reference trajectory. For this system the data was obtained by using both Tustin approximation and the usual zero-order hold. The comparison of these simulations is in Figure 3-8.

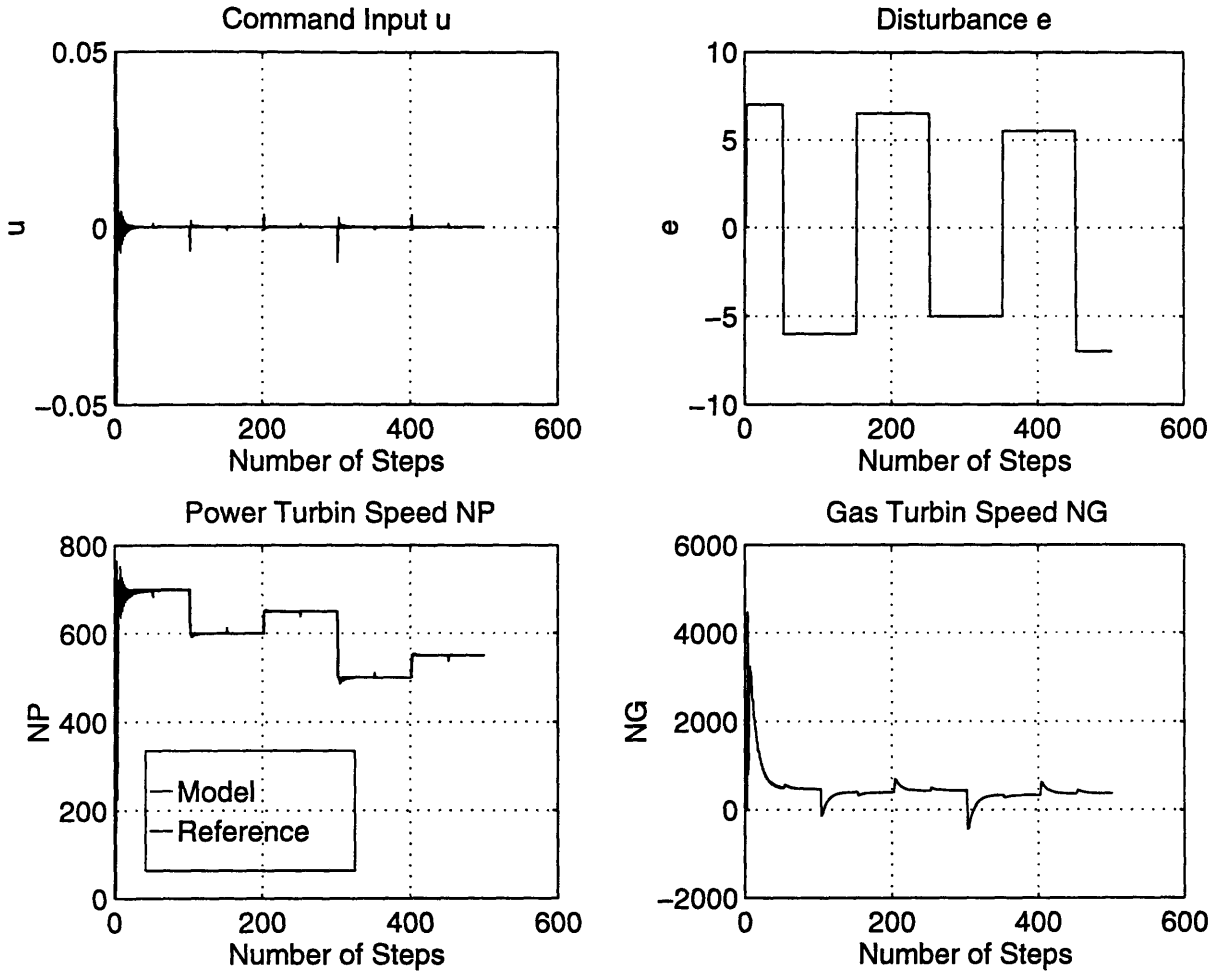

Figure 3-5: Self-tuning simulation output of reduced order model of T700 engine

3.3.4 A Two Input-Two Output System

In this simulation, a system with two outputs and two inputs was examined[5]. The system has a transfer function matrix in the form:

$$\mathrm{G}(s) = \frac{e^{-\Delta s}}{s^2 + 3s - 1} \begin{pmatrix} s+1 & 3 \\ 1 & s+2 \end{pmatrix} \tag{3.40}$$

The following data was assumed for the system: A delay of 0.015 seconds was assumed, h=0.01 sec., Q_1=1, Q_2=0.002, saturation values of the input command was taken to be -8 and 8 respectively, β=0.98, y_0=0 and the reference trajectory was taken to be

[5]This system was taken and adapted from [51].

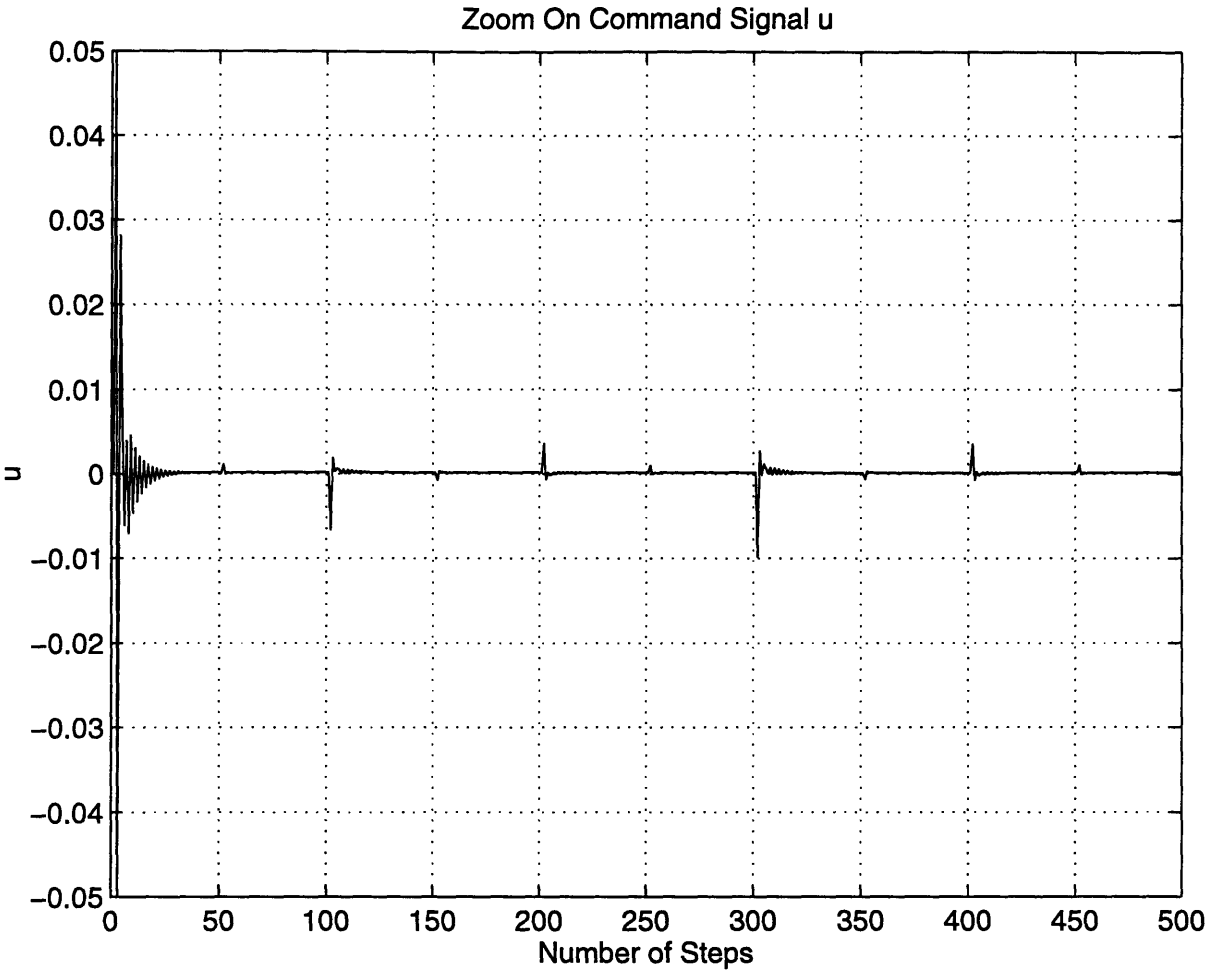

Figure 3-6: Detailed plot of command signal u

y_1=1, and y_2=1.5. For the identification routine, $n = 3$ and $m = 1$ was selected. Then the self-tuning script was prepared (A.4.5). The results of this simulation can be seen in Figure 3-9. This figure shows the outputs together with the necessary command input values. As we can see from the figure, the results are acceptable.

3.4 System Identification and Simulations

We have seen in the previous sections that identification[6] of the dynamic system, as in any other control strategy, is one of the most important parts of self-tuning control. If the system is modelled appropriately, the user can depend on the results

[6]For a list and simple explanations of common identification schemes refer to Appendix B.

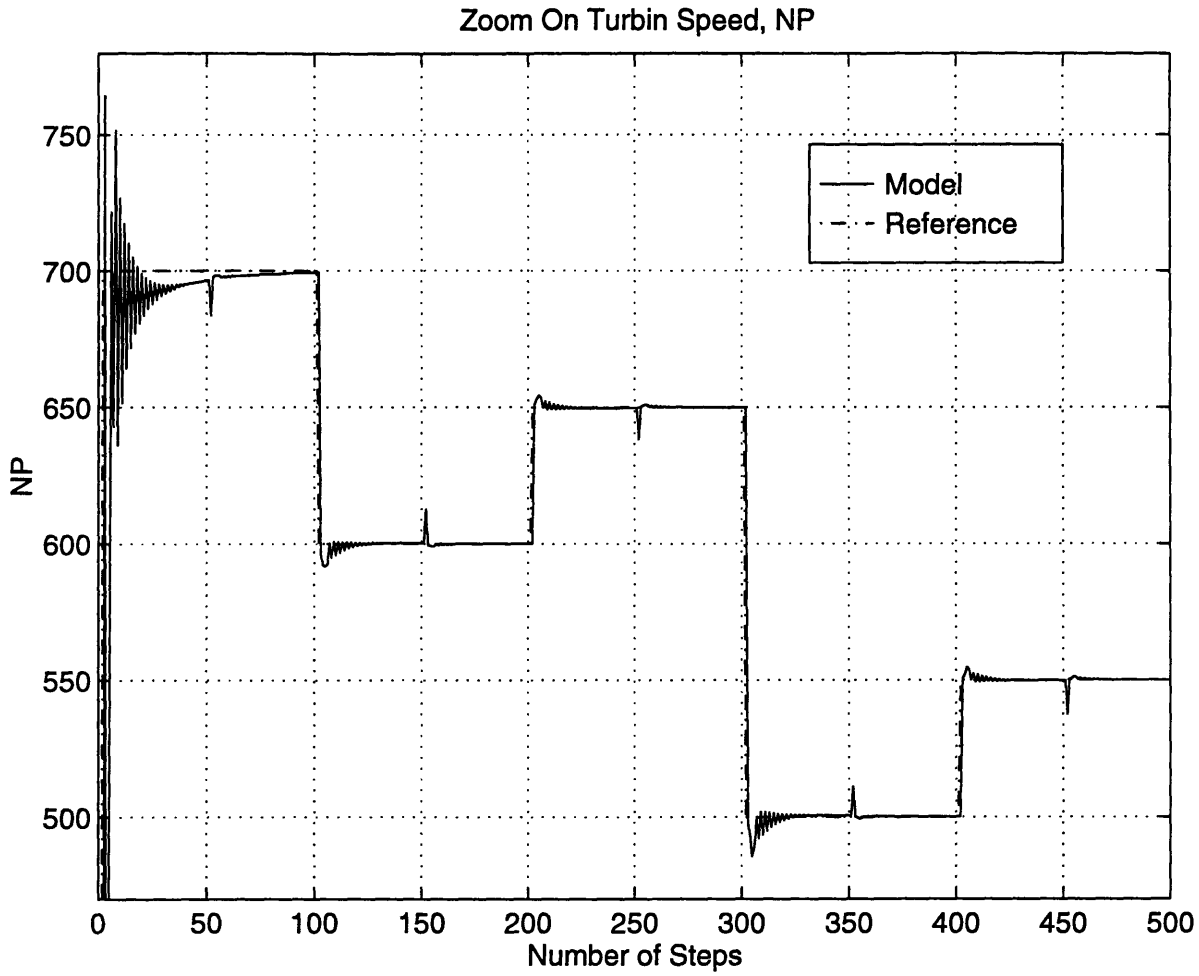

Figure 3-7: Detailed plot of power turbine speed (controlled parameter), NP

of simulations. Otherwise, the simulations can be meaningless.

Matlab^{TM} has a special toolbox, called the System Identification Toolbox, for this purpose. In this toolbox the user can use Least Squares to build an ARX (**A**uto **R**egressive with e**X**ogeneous inputs) model, use Instrumental Variable Method (IV4), ARMAX model, Box-Jenkins model. All of these methods are very useful in particular applications.

In [30] and [40] the discreperancies between the real system and the model were discussed. By using the main ideas of these articles and by investigating the script files of Matlab^{TM}, in this thesis, a comparably short identification script was prepared. The script can be found in Appendix 4.6. Besides this script file, a set of script files, to pursue ARX modeling using Least Squares, was also prepared (see Appendix

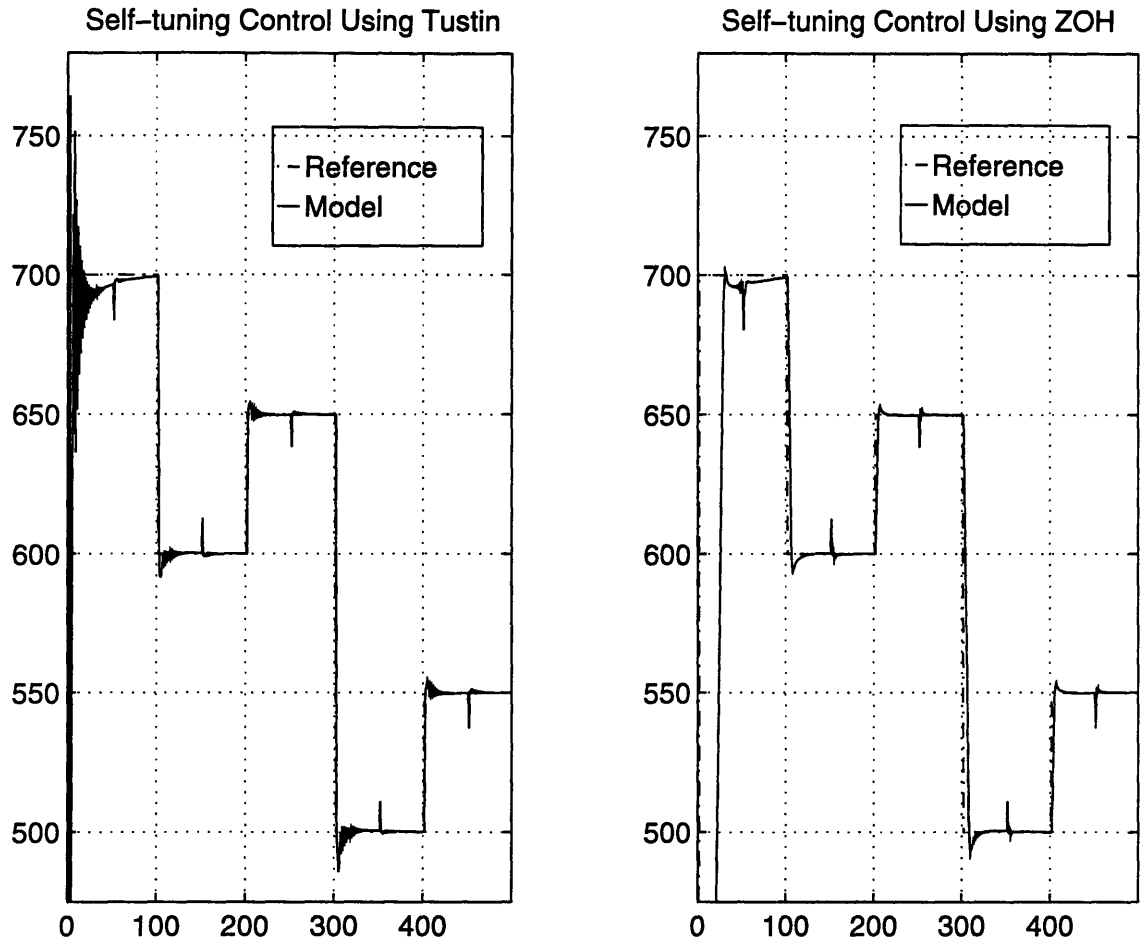

Figure 3-8: Comparison of the Simulations with Tustin Approximation and ordinary zero order hold

5). In the next section the simulations made with these identification scripts will be introduced.

3.4.1 Identification Of an SISO System

The first system that was identified was the same system given by equation (3.1). The system was assumed to have a delay of 0.015 seconds. The identification program was run assuming $n = 2$, $m = 1$ and $k = 1$. The results of this simulation can be seen in figures 3-10 and 3-11[7]. The first figure shows the comparison of the model to the

[7]Here the step input result of the real system is found by integrating the impulse response (cumulative sum).

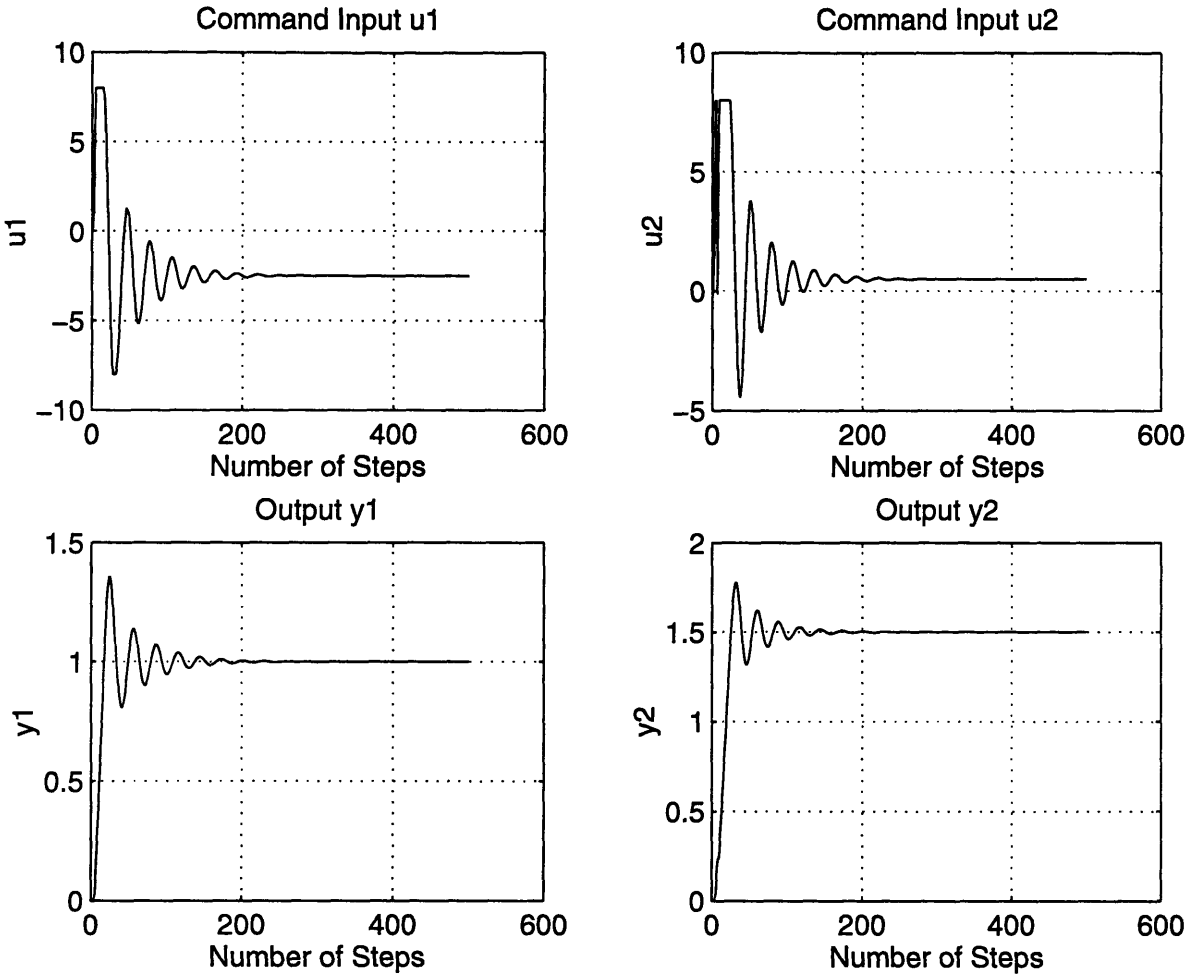

Figure 3-9: Simulation results of a Multi Input-Multi Output(MIMO) System

real system, when a random input is applied. The second one shows the comparison of the model and the real system when a step input is applied. As it can be seen from the plots, the loss that is calculated by Matlab^{TM} is fairly small, and the model approaches the real system.

The same script was also run with different transfer functions, and in all times very reasonable results were obtained.

After successfully completing several simulations with this particular script file, a new set of identification scripts were prepared, and with this new set several more simulations accomplished. It was explained before that this set uses Least Squares estimation to build an ARX model. It also calculates the loss with a loss function in

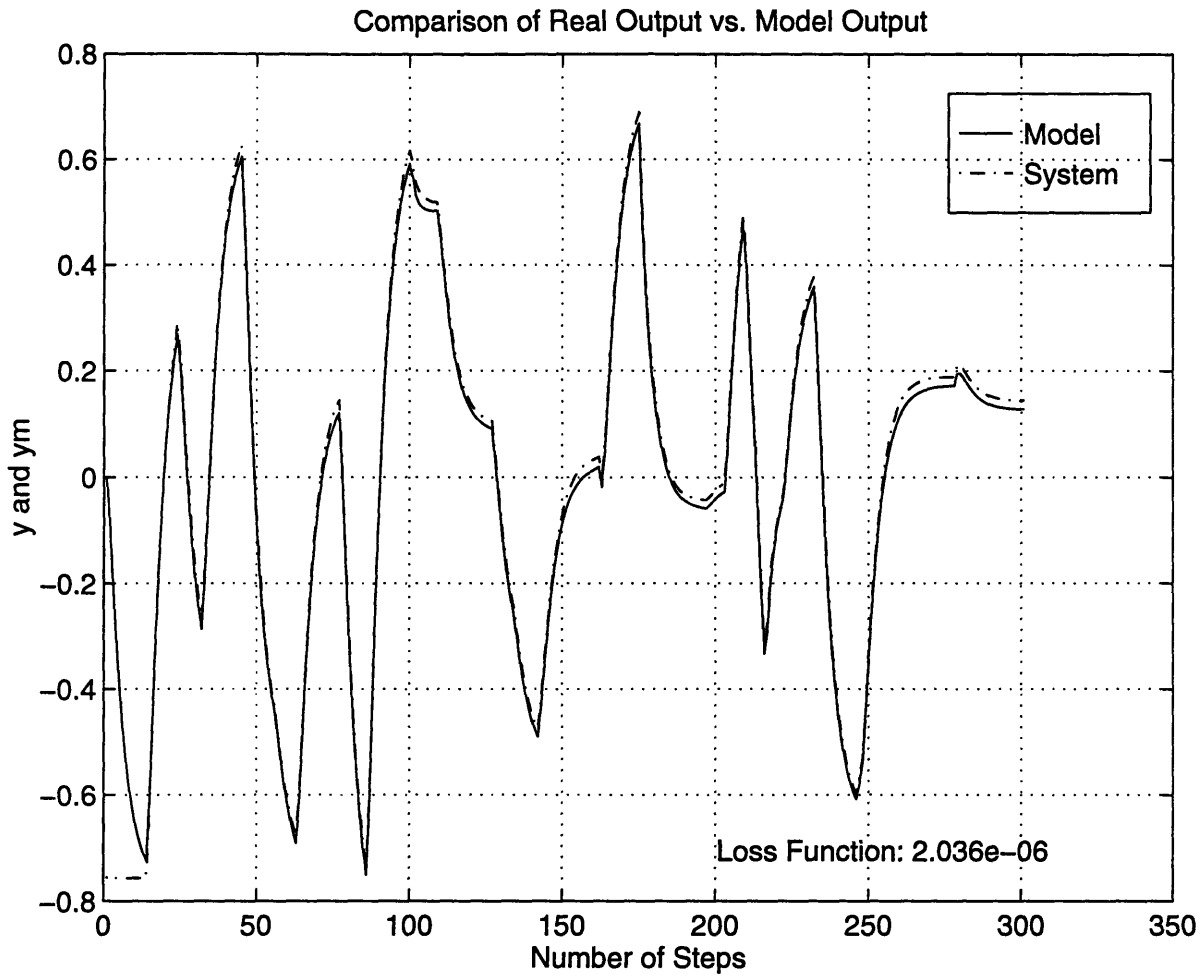

Figure 3-10: Identification Result of An SISO system using simpler version of system identification toolbox of Matlab^{TM}

the form:

$$V = \sum_{i=1}^{n} (y(i) - y_m(i))^2 \tag{3.41}$$

where $y(t)$ is the output of the system and $y_m(t)$ is the output of the model.

The identification of the system that was done using this script set had the transfer function in the form:

$$G(s) = \frac{1}{(9.375\, s^2 + 6.25\, s + 1)} \tag{3.42}$$

The identification script can be seen in A.5.4. For the model, it was assumed that $n = 2$, $m = 2$, $d = 0$. (The order of the model was selected different from the real system, intentionally, to see how well the the scripts work.) The results of this identification, together with step input responses can be seen in figures 3-

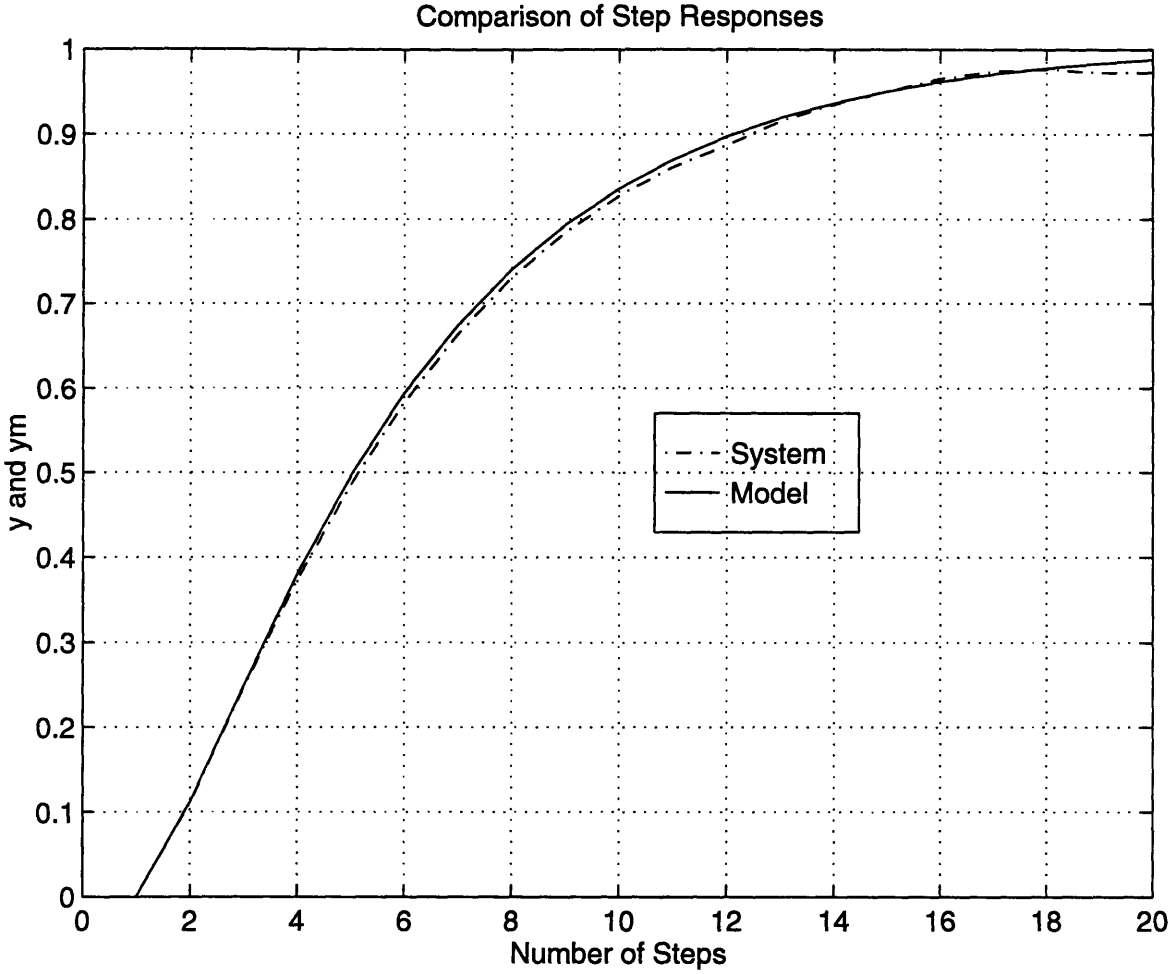

Figure 3-11: Comparison of Step Input Results of the model and the real system

12 and 3-13[8]. As it can be seen from these figures, the results are comparable to Matlab^{TM}'s identification toolbox. And the good thing is that the parameter vector is very convenient to use in the general self-tuning control scripts that were prepared and discussed earlier.

In fact, the same self-tuning control simulations that were described in section 3.3 was redone by using this new script set, and the same results were obtained.

In this chapter, the examples and applications of the general Minimum Variance self-tuning control program and system identification using least squares estimation that was prepared for this thesis were given. The results indicate that the programs

[8]The simulation of the model is by using the dlsim (digital simulation) command.

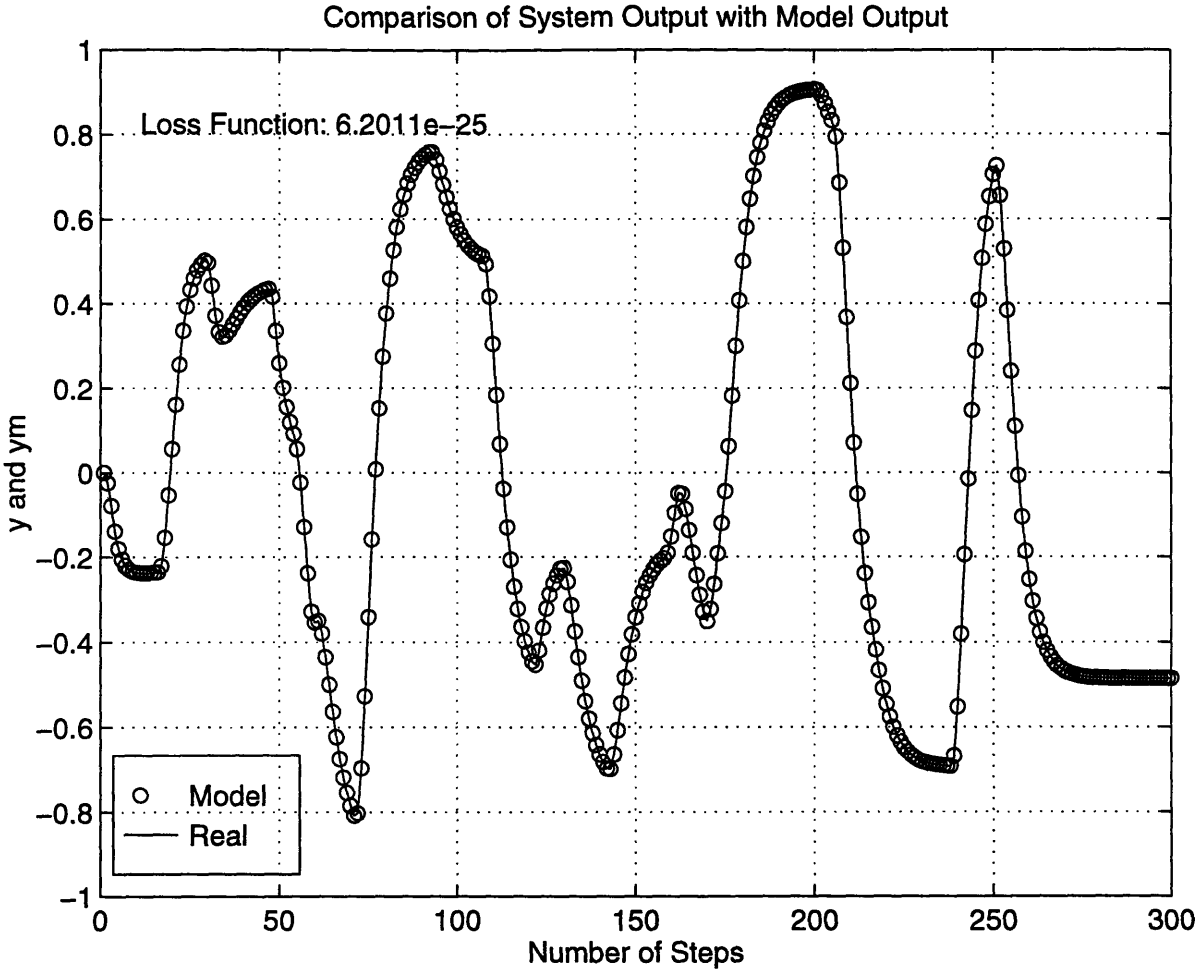

Figure 3-12: Identification result of a system using new scripts

are producing sufficient responses and can be used for any dynamic system[9].

[9]The examples that were included are minimum phase systems. For non-minimum phase systems, they can still be applied (in fact they were applied). But for better applications self-tuning algorithm can be modified. For further discussion the reader should refer to [65].

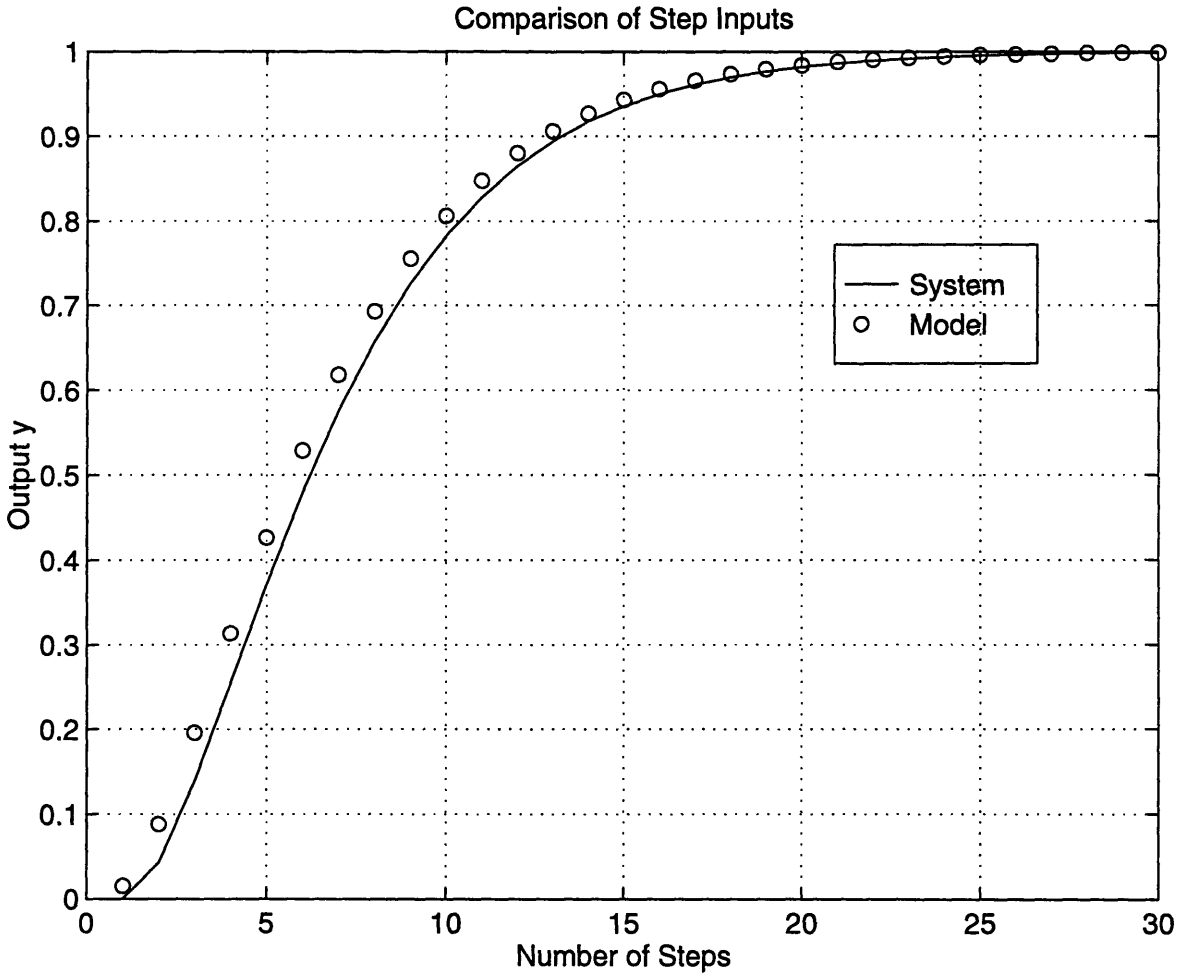

Figure 3-13: Comparison of Step Input Results of the model and the real system using new scripts

Chapter 4

Results and Conclusion

In the last years, self-tuning control grew from a theoretical subject to an important industrial tool. Practical applications are being established more and more, and it appears that the methods are becoming a standard part of a control engineer's tools. In the acceptance of self-tuning, microprocessors played a very important role because of their capability for the computationally demanding algorithms to be implemented inexpensively.

As we have seen throughout our simulations and applications, self-tuners do not eliminate the need for an engineer's skill but allows him to think better about the real control needs and constraints of the plant at hand instead of simply hoping that manually-tuned and fixed PID laws will solve all of his problems [14].

Today, the most important thing is to add newly developed engineering features to the self-tuning methods. As we have seen throughout this thesis, self-tuning mainly depends on the process model that we built from input/output data. This identification part becomes more and more difficult if we have actuator and sensor nonlinearities, unmodelled dynamics, and certain types of disturbances. In order to supply a suitable model in this case, Gawthrop [14] suggests to provide recursive parameter estimation methods which are reliable: often called "jacketing" software must be added to detect periods of poor data.

A second point, the self-tuning control needs to be computerized. Reliable and reusable subroutines or codes that clearly show the appropriate algorithms and data

structures should be prepared. In fact this part was the main topic of this thesis. As we can see from the previous chapter, the scripts included in this thesis are easy to understand and implement. They were prepared for Minimum Variance Control, but any other self-tuning algorithm (such as generalized minimum variance or pole placement control) can be added easily. On adding such algorithms we can also create a small test routine and let the program choose the best way itself.

Another issue that can be done with these scripts is the optimal working conditions. As we have seen the weight factors are constant, but this, sometimes, can cause non-optimal conditions for the process. In order to eliminate this unwanted situation, one can try to make the factors change at each simulation step.

Appendix A

Matlab^{TM} Program Files

A.1 Parameter Estimation Program for a Well-known System

```
% An .m file that makes parameter estimation and simulation for a
% system in the form:
%
% Y(s)/U(s) = e^(-Δ s) (τ1 s+1)/((τ2 s+1)(τ3 s+1))
% Y(s)/U(s)=(e^(-delta*s))*(to1*s+1)/((to2*s+1)*(to3*s+1))
%
clear
to1=0.1;
to2=0.2;
to3=0.3;                                                    10
delta=.015; % Time delay.
h=.01; % Time interval.
W=0.98; % Weighing factor in RLS algorithm.
rs=1e4; % Starting value of matrix R.
kson=500;
us=1;
d=fix(delta/h)+1;
num=[to1 1];
```

```
den=[to2*to3 to2+to3 1];
disp('Y(s)/U(s)=')
printsys(num,den)
[A,B,C,D]=tf2ss(num,den); % Transfer function to state space.
[Aa,Bb,Cc,Dd]=c2dt(A,B,C,h,delta); % Continuous to discrete with delay.
[numz,denz]=ss2tf(Aa,Bb,Cc,Dd);
disp('Y(z)/U(z)=')
printsys(numz,denz,'z')
%
[u,ub]=uinput(kson,us);
y=dlsim(Aa,Bb,Cc,Dd,u); % Simulation.
plot(y)
R=eye(5)*rs; % R is diagonal in the beginning.
P(3,:)=[1,0,0,0,0];
q=kson-4+d;
for i=d+1:q;
fi=[u(i+2-d),u(i+1-d),u(i-d),-y(i+1),-y(i)]'; % i+2 , Present time.
% RLS
S=W+fi'*R*fi;
R=(R-(R*fi*fi'*R/S))/W;
P(i+1,:)=P(i,:)+(y(i+2)-P(i,:)*fi)*fi'*R;
end
subplot(2,3,1);plot(P(:,1));title('b0')
subplot(2,3,2);plot(P(:,2));title('b1')
subplot(2,3,3);plot(P(:,3));title('b2')
subplot(2,3,4);plot(P(:,4));title('a1')
subplot(2,3,5);plot(P(:,5));title('a2')
disp('Please presss <ENTER> after you see the graphics:')
pause
%
yy=dlsim(Aa,Bb,Cc,Dd,ub);
subplot(2,1,1)
plot(yy,'c');title('Simulation with real values')
axis([0 kson 0 us+.5])
numr=[P(kson-2,1) P(kson-2,2) P(kson-2,3)];
denr=[1 P(kson-2,4) P(kson-2,5) zeros(1,d)];
```

```
disp('Y(z)/U(z)=    that we get with estimation')
printsys(numr,denr,'z')
yr=dlsim(numr,denr,ub);
subplot(2,1,2);plot(yr);title('Simulation with estimation values')
```

A.1.1 Input-signal File for Parameter Estimation Program

```
% This is a Matlab file that produces random command input (u).
% u and ub are the output of the file; kson and us are the input parameters.
% u   : Random command input (will be used in parameter estimation).
% ub  : Unit input command (will be used in simulation).
% kson: Step number.
% us  : Command input width.
function [u,ub]=uinput(kson,us)
u1=2*rand*us-1;
for i=1:kson;
  l=2*rand-1.8;                                                    10
    if l>0
     u1=2*rand*us-1;
    end
  u(i)=u1;
  ub(i)=us;
end
u=u';
ub=ub';
```

A.2 General Parameter Estimation Program

```
function [P]=genest(u,y,m,n,d)
disp('GENERAL SYSTEM PARAMETER ESTIMATION')
%
disp(' ')
disp('Y(z^-1)     z^-d*(B0+B1*z^-1+...+Bm*z^-m)')
disp('------- = ------------------------------')
disp('U(z^-1)     I+A1*z^-1+A2*z^-2+...+An*z^-n')
disp(' ')
ts=size(y,1);
k=size(y,2);                                                    10
l=size(u,2);
dmn=(m+1)*l+n*k;
W=0.98;
R=eye(dmn)*10^4;
P=[eye(k) zeros(k,dmn-k)];
%
t0=d+m+1;
if n>(m+d)
 t0=n+1;
end                                                             20
% RLS
for t=t0:ts;
fi=u(t-d,:)';
for i=1:m;
 fi=[fi;u(t-d-i,:)'];
end
for j=1:n;
 fi=[fi;-y(t-j,:)'];
end
S=W+fi'*R*fi;                                                   30
R=(R-(R*fi*fi'*R)/S)/W;
P=P+(y(t,:)'-P*fi)*fi'*R;
end
%
```

```
disp('Coefficients of polynomial B:')
disp('========================')
for i=0:m;
 disp(P(:,l*i+1:l*(i+1)))
 disp('________________________')
end
disp('Coefficients of polynomial A:')
disp('========================')
for j=1:n;
 disp(P(:,(m+1)*l+k*(j-1)+1:(m+1)*l+k*j))
 disp('________________________')
end
% End of file.
```

A.2.1 Least-squares Parameter Estimation Program

```
% This program is just to find the coefficients of our model
% using Least-Squares Parameter Estimation. Inputs are n,m,d
% and our real input & output signals from the system.

t=size(u,1);
ym=y(n+1:t,:);
fi=[];
for i=1:n;fi=[fi -y(n+1-i:t-i,:)];end
for j=0:m;fi=[fi u(m+d-j:t-n+m+d-1-j,:)];end
P=(inv(fi'*fi)*fi'*ym)';
```

A.3 General Minimum Variance Self-tuning Control Program

```
function [u,e,y,P]=genstc(model,AR,BR,CR,nu,y0,yr,d,st,mn,RI,W,QQ1,QQ2,umin,umax)
% GENERAL MINIMUM-VARIANCE SELF-TUNING CONTROL
%
%
% If you want to give a state space model for your real system, use:
% [u,e,y,P]=genstc('cm',AC,BC,CC,DC,y0,yr,h,std,mean,RI,W,QQ1,QQ2,umin,umax)
% *******
% If you want to give a time series model for your real system, use:
% [u,e,y,P]=genstc('tm',AR,BR,CR,nu,y0,yr,d,std,mean,RI,W,QQ1,QQ2,umin,umax)
%                                                                          10
%
% Parameters in the following expression will be estimated:
% (I+A1*z^-1+...+An*z^-n)*y(z^-1)=z^-d*(B0+B1*z^-1+...+Bm*z^-m)*u(z^-1)+
%                                 +(I+c1*z^-1+...+Ck*z^-k)*e(z^-1)
% A(q)*y(t)=B(q)*u(t-d)+C(q)*e(t)
%
%       u,e,y: input,white-noise and output matrices
%          P: P=[B0 B1..Bm A1 A2..An C1 C2..Ck]: estimated parameters
%          h: sampling period
%          d: delay in discrete time                                       20
%         nu: numbers of inputs
% AC,BC,CC,DC: continuous model matrices for real system
%   AR,BR,CR: time series model matrix polynomials for real system
%              AR=[I A1...An] , BR=[B0 B1...Bm] , CR=[I C1...Ck]
%         y0: y0=[y1(0) y2(0)...]: initial conditions for outputs
%         yr: yr=[yr(1);yr(2);...;yr(t)]: reference (desired output) matrix
%   std,mean: standard deviation and mean for white noise
%         RI: initial value of R. (not a matrix - only one element)
%          W: weighting factor for RLS
% QQ1 and QQ2: weights (not matrices) to compute input u                   30
% umin, umax: restriction values on input u
%
```

```
ts=size(yr,1);ny=size(yr,2);ne=ny;
if model=='cm';h=d;dl=input('delay(sec)=?');d=fix(dl/h)+1;
[AD,BD,CD,DD]=c2dm(AR,BR,CR,nu,h,'zoh');
% or [AD,BD,CD,DD]=c2dm(AR,BR,CR,nu,h,'tustin');
nu=size(BR,2);
disp('(I+A1*z^-1+...+An*z^-n)*y(z^-1)=z^-d*(B0+B1*z^-1+...+Bm*z^-m)*u(z^-1)+')
disp('                                    +(I+c1*z^-1+...+Ck*z^-k)*e(z^-1)')
n=input('n:  degree of polynomial A=?');                                    40
m=input('m:  degree of polynomial B=?');
k=input('k:  degree of polynomial C=?');
nx=size(AD,2);end
if model=='tm';n=size(AR,2)/ny-1;m=size(BR,2)/nu-1;k=size(CR,2)/ne-1;
PR=[BR AR(:,ny+1:ny*(n+1)) CR(:,ne+1:ne*(k+1))];end
dmn=nu*(m+1)+n*ny+k*ne;
R=eye(dmn)*RI;Q1=eye(ny)*QQ1;Q2=eye(nu)*QQ2;
P=[eye(ny) zeros(ny,dmn-ny)];
%
t0=d+m+1;                                                                   50
if n>(m+d);t0=n+1;end
te=ts+t0-1;yr=[zeros(t0-1,ny);yr];y=zeros(te+d,ny);
u=zeros(te+d,nu);e=zeros(te+d,ne);
y(t0-1,:)=y0;
if model=='cm';x(t0-1,:)=(CD'*inv(CD*CD')*y0')';end
% Starting of main loop
for t=t0:te;
%
% Computing white-noise
for i=1:ne                                                                  60
e(t,i)=wnoise(st,mn);
end
    %
    % Real system output
    if model=='cm'
      x(t,:)=(AD*x(t-1,:)'+BD*u(t-d,:)')';
      y(t,:)=(CD*x(t,:)'+DD*u(t-1,:)')'+e(t,:);
    end
```

```
    if model=='tm';fi=u(t-d,:)';
    for i=1:m;fi=[fi;u(t-d-i,:)'];end                                    70
    for j=1:n;fi=[fi;-y(t-j,:)'];end
    for l=1:k;fi=[fi;e(t-j,:)'];end
    y(t,:)=(PR*fi)'+e(t,:);end
    %
[R,P,u]=ucrstc(t,nu,ny,ne,m,n,k,yr,u,e,y,d,P,R,W,Q1,Q2,umin,umax);
end
% Rearranging u,y and e
u=u(t0-1:ts+t0-2,:);y=y(t0-1:ts+t0-2,:);e=e(t0-1:ts+t0-2,:);
% End of file
```

A.3.1 Input Creating Program

```
function [R,P,u]=ucrstc(t,nu,ny,ne,m,n,k,yr,u,e,y,d,P,R,W,Q1,Q2,umin,umax)
% GENERAL MINIMUM-VARIANCE SELF-TUNING CONTROL TO CREATE INPUT U
%
%
% [R,P,u]=ucrstc(t,nu,ny,ne,m,n,k,yr,u,e,y,d,P,R,W,Q1,Q2,umin,umax)
%
% Parameters in the following expression will be estimated:
% (I+A1*z^-1+...+An*z^-n)*y(z^-1)=z^-d*(B0+B1*z^-1+...+Bm*z^-m)*u(z^-1)+
%                                 +(I+c1*z^-1+...+Ck*z^-k)*e(z^-1)
% A(q)*y(t)=B(q)*u(t-d)+C(q)*e(t)                                          10
%
%          t: discrete time (step)
%       u,e,y: input,white-noise and output matrices
%          P: P=[B0 B1..Bm A1 A2..An C1 C2..Ck]: estimated parameters
%       m,n,k: degrees of polynomial B,A and C
%          d: delay in discrete time
%    nu,ny,ne: numbers of inputs,outputs and disturbances
%          yr: yr=[yr(1);yr(2);...;yr(t)]: reference (desired output) matrix
%          R: a variable for RLS
%          W: weighting factor for RLS                                     20
%       Q1,Q2: weights to compute input u
%   umin,umax: restriction values on input u
%
nn=nu*(m+1);mm=nn+n*ny;
fi=u(t-d,:)';
for i=1:m;fi=[fi;u(t-d-i,:)'];end
for j=1:n;fi=[fi;-y(t-j,:)'];end
for l=1:k;fi=[fi;e(t-j,:)'];end
% RLS
S=W+fi'*R*fi;R=(R-(R*fi*fi'*R)/S)/W;P=P+(y(t,:)'-P*fi)*fi'*R;              30
%
CU=zeros(ny,1);
for j=1:m;CU=CU+P(:,nu*j+1:nu*(j+1))*u(t-j,:)';end
%
```

```
  % Computing future values of y when necessary and using them to compute CU
  for ii=1:d-1;fi1=u(t-d+ii,:)';
  for i=1:m;fi1=[fi1;u(t-d-i+ii,:)'];end
  for j=1:n;fi1=[fi1;-y(t-j+ii,:)'];end
  for l=1:k;fi1=[fi1;e(t-l+ii,:)'];end
  y(t+ii,:)=(P*fi1)';                                                    40
  p1=nn+ny*(d-ii-1)+1;p2=p1+ny-1;
  if (d-ii)<(n+1);CU=CU-P(:,p1:p2)*y(t+ii,:)';end
  end
  %
%
for jj=d:n;CU=CU-P(:,nn+ny*(jj-1)+1:nn+ny*jj)*y(t-jj+d,:)';end
for kk=d:k;CU=CU+P(:,mm+ne*(kk-1)+1:mm+ne*kk)*e(t-kk+d,:)';end
CU=CU-yr(t,:)';
% Computing input u
u(t,:)=(-inv(P(:,1:nu)'*Q1*P(:,1:nu)+Q2)*(P(:,1:nu)'*Q1*CU-(u(t-1,:)*Q2)'))';   50
                                                  Integral action
  % Saturation
  for i=1:nu;
  if u(t,i)>umax;u(t,i)=umax;end
  if u(t,i)<umin;u(t,i)=umin;end
  end
% End of file.
```

A.3.2 White Noise Program

```
function ewn=wnoise(std,mean)
% WHITE-NOISE FILE
%
%
% [R,P,u]=ucrstc(t,nu,ny,ne,m,n,k,yr,u,e,y,d,P,R,W,Q1,Q2,umin,umax)
%
% ewn : white-noise (only one number - not a matrix)
% std : standard deviation
% mean: mean
%                                                                    10
s=-6;
for i=1:12;
s=s+rand;
end
ewn=std*s+mean;
% end of file
```

A.4 Demonstration Programs

```
% GENERAL SELF TUNING CONTROL DEMOS
% AUTHOR: Ali Yurdun ORBAK  -  Massachusetts Institute of Technology.
%
labels = str2mat(...
      'Continuous Model Demo',...
      'Time-series Model Demo',...
      'Demo w/your Special file',...
      'T700 Engine Demo',...
      'System-ID Demo');
% Callbacks                                                          10
callbacks =[ ...
        'stcdemo1  '
        'stcdemo2  '
        'stcdemo3  '
        'stcdemo4  '
        'idmodel1  '];

yurfl2(labels,callbacks);
```

A.4.1 Program For Creating Demo Window

```
function yurfl2(labelList,nameList,figureFlagList);
% YURDUNFL2 An Expo gateway routine for playing self-tuning control demos.

%
% GENERAL SELF-TUNING CONTROL DEMOS WINDOW
% AUTHOR: Ali Yurdun Orbak - 11-20-1994 Massachusetts Institite of Technology
%       Note: Prepared by using cmdlnwin.m file in Matlab4.2 (c).
%             This file is a changed and adapted version of cmdlnwin.m file.
%             This script is prepared for the demonstrations of this thesis.
%                                                                          10

% labelList contains the descriptive names of the demos
% nameList contains the actual function names of the demos
% windowFlagList contains a flag variable that indicates whether
% or not a window is required for the demo

oldFigNumber=watchon;

% If no figureFlagList is supplied, assume every demo needs a
% figure window                                                            20

if nargin<3,
   figureFlagList=ones(size(labelList,1),1);
end

% Now initialize the whole figure...

figNumber=figure( ...
   'Name','Self-tuning Control Demos by Ali Yurdun Orbak', ...
   'NextPlot','New', ...                                                 30
   'NumberTitle','off');

axes('Visible','off', ...
   'NextPlot','new')
```

```
%==================================
% Set up the Comment Window
top=0.30;
left=0.05;
right=0.75;                                                    40
bottom=0.05;
labelHt=0.05;
spacing=0.005;
promptStr= ...
  [' The buttons in this window will launch "Self-Tuning  '
      ' Control Demos". These demos  use the MATLAB        '
      ' Command Window for input and output. Make sure the  '
      ' command window is visible before you run these demos.'];

% First, the Comment Window frame                              50

frmBorder=0.02;
frmPos=[left-frmBorder bottom-frmBorder ...
      (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
uicontrol( ...
   'Style','frame', ...
   'Units','normalized', ...
   'Position',frmPos, ...
      'BackgroundColor',[0.50 0.50 0.50]);
                                                               60
% Then the text label

labelPos=[left top-labelHt (right-left) labelHt];
uicontrol( ...
      'Style','text', ...
   'Units','normalized', ...
   'Position',labelPos, ...
   'BackgroundColor',[0.50 0.50 0.50], ...
      'ForegroundColor',[1 1 1], ...
   'String','Comment Window');                                 70
```

```
   % Then the editable text field

   txtPos=[left bottom (right-left) top-bottom-labelHt-spacing];
   txtHndl=uicontrol( ...
         'Style','edit', ...
      'Units','normalized', ...
      'Max',10, ...
      'BackgroundColor',[1 1 1], ...
      'Position',txtPos, ...                                          80
      'String',promptStr);

%=====================================
% Information for all buttons

labelColor=[0.8 0.8 0.8];
yInitPos=0.90;
top=0.95;
bottom=0.05;
left=0.80;                                                            90
btnWid=0.15;
btnHt=0.10;
% Spacing between the button and the next command's label
spacing=0.04;

%=====================================

% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;                                                  100
frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
h=uicontrol( ...
   'Style','frame', ...
   'Units','normalized', ...
   'Position',frmPos, ...
   'BackgroundColor',[0.5 0.5 0.5]);
```

```
%====================================

% The INFO button                                                    110
labelStr='Info';

infoStr= ...
   ['                                  '
    ' The "Self-tuning Control Demos" window     '
    ' allows you to launch demos that operate    '
    ' from the  command window, as opposed to    '
    ' those that have their own graphic user     '
    ' interface.                                 '
    ' This allows users of The MATLAB Expo to    '                  120
    ' have access to a number of demos that, for '
    ' one reason or another, do not specifically '
    ' use the GUI tools.                         '
    ' Remember that these demos will send output '
    ' to and accept input from the MATLAB        '
    ' command window, so you must make sure      '
    ' that the command window does not get hidden'
    ' behind any other windows.                  '
    ' These Demos are written by Ali Yurdun Orbak'
    ' for Thesis at MIT. (Cuneyt Yilmaz from     '                  130
    ' ITU also helped for some of these demos.   '
    ' File name: yurfl2.m                        '];

callbackStr='helpfun(''Self-tuning Control Demo Info'',get(gco,''UserData''))';
infoHndl=uicontrol( ...
   'Style','pushbutton', ...
   'Units','normalized', ...
   'Position',[left bottom+btnHt+spacing btnWid btnHt], ...
   'String',labelStr, ...
   'UserData',infoStr, ...                                            140
   'Callback',callbackStr);
% ====================================
```

```
% The STOP button
%   labelStr='Stop';
%   callbackStr='set(gcf);';
%
%   % Generic  button information
%   btnPos=[left bottom+2*(btnHt+spacing)  btnWid btnHt];
%   stopHndl=uicontrol( ...                                          150
%       'Style','pushbutton', ...
%       'Units','normalized', ...
%       'Position',btnPos, ...
%       'Enable','on', ...
%       'String',labelStr, ...
%       'Callback',callbackStr);

%=====================================

% The CLOSE button                                                  160
labelStr='Close';
callbackStr='close(gcf)';
closeHndl=uicontrol( ...
   'Style','pushbutton', ...
   'Units','normalized', ...
   'Position',[left bottom btnWid btnHt], ...
   'String',labelStr, ...
   'Callback',callbackStr);

%=====================================                             170
% Information for demo buttons

labelColor=[0.8 0.8 0.8];
btnWid=0.33;
btnHt=0.08;
top=0.95;
bottom=0.35;
right=0.75;
```

```
leftCol1=0.05;
leftCol2=right-btnWid;                                                    180
% Spacing between the buttons
spacing=0.02;
%spacing=(top-bottom-4*btnHt)/3;

numButtons=size(labelList,1);
col1Count=fix(numButtons/2)+rem(numButtons,2);
col2Count=fix(numButtons/2);

% Lay out the buttons in two columns
                                                                          190
for count=1:col1Count,

   btnNumber=count;
   yPos=top-(btnNumber-1)*(btnHt+spacing);
   labelStr=deblank(labelList(count,:));
   callbackStr='eval(get(gco,''UserData''));';
   cmdStr=['figureNeededFlag=',num2str(figureFlagList(count)),'; ', ...
         'cmdlnbgn; ' nameList(count,:) '; cmdlnend;'];

   % Generic button information                                           200

   btnPos=[leftCol1 yPos-btnHt btnWid btnHt];
   startHndl=uicontrol( ...
      'Style','pushbutton', ...
      'Units','normalized', ...
      'Position',btnPos, ...
      'String',labelStr, ...
         'UserData',cmdStr, ...
      'Callback',callbackStr);
end;                                                                      210

for count=1:col2Count,

   btnNumber=count;
```

```
    yPos=top-(btnNumber-1)*(btnHt+spacing);
    labelStr=deblank(labelList(count+col1Count,:));
    callbackStr='eval(get(gco,''UserData''));';
    cmdStr=['figureNeededFlag=',num2str(figureFlagList(count+col1Count)),'; ', ...
            'cmdlnbgn; ' nameList(count+col1Count,:) '; cmdlnend;'];
                                                                              220
    % Generic button information

    btnPos=[leftCol2 yPos-btnHt btnWid btnHt];
    startHndl=uicontrol( ...
       'Style','pushbutton', ...
       'Units','normalized', ...
       'Position',btnPos, ...
       'String',labelStr, ...
          'UserData',cmdStr, ...
       'Callback',callbackStr);                                               230

end;

watchoff(oldFigNumber);
```

A.4.2 Demo-1

```
% DEMO-1 ABOUT SELF TUNING CONTROL
%
%
clear;clf;
num=[0.1 1];den=[0.06 0.5 1];
disp('System:  SISO       .')
disp('                    x=AC*x+BC*u')
disp('                    y=CC*x+DC*u')
[AC,BC,CC,DC]=tf2ss(num,den)
disp('Press any key to continue.')                                    10
pause
clc
for i=1:100;yr(i,:)=1;end
for i=101:200;yr(i,:)=-1;end
y0=0;
h=0.01;
std=0.001;
mean=0;
RI=1e4;
W=0.98;                                                               20
QQ1=1;QQ2=0.001;
umin=-10;umax=10;
disp('yr will be 1 for 100 steps, and then -1 for 100 steps.')
disp('y0=0  h=0.01 sec  std=0.001   mean=0   RI=1E4   W=0.98')
disp('QQ1=1   QQ2=0.001   umin=-50   umax=50  will be taken.')
disp('enter:  delay=0.015 sec   n=3   m=2  k=0')
[u,e,y,P]=genstc('cm',AC,BC,CC,DC,y0,yr,h,std,mean,RI,W,QQ1,QQ2,umin,umax);
subplot(2,1,1);plot(u);grid;title('Command Input u');
subplot(2,1,2);plot(y);grid;title('Output y');
end                                                                   30
% end of demo-1.
```

A.4.3 Demo-2

```
% DEMO-2 ABOUT SELF TUNING CONTROL
%
%
clear
disp('System:  a two input-two output system')
disp('(I+A1*z^-1+A2*z^-2)*y(z^-1)=(B0+B1*z^-1)*u(z^-1)+e(z^-1)')
A1=[-1.5 0.3;0.2 -1.5]
A2=[0.54 -0.1;0.1 0.56]
B0=[2 -0.3;0.1 0.56]
B1=[-1.8 0.2;-0.2 -0.5]                                          10
AR=[eye(2) A1 A2];
BR=[B0 B1];
CR=eye(2);
disp('press any key to continue.')
pause
clc
for i=1:100;yr(i,1:2)=[1 1];end
y0=[0 0];
nu=2;
d=1;                                                             20
RI=1E+04;
W=0.98;QQ1=1;QQ2=0.001;
std=0.01;mean=0;
umin=-50;umax=50;
disp('yr will be [1 1] for 100 steps.')
disp('y0=[0 0] nu=2  d=1  RI=1E4  W=0.98  QQ1=1  QQ2=0.001')
disp('std=0.01  mean=0  umin=-50  umin=50  will be taken.')
[u,e,y,P]=genstc('tm',AR,BR,CR,nu,y0,yr,d,std,mean,RI,W,QQ1,QQ2,umin,umax);
subplot(2,2,1);plot(u(:,1));title('u1')
subplot(2,2,2);plot(u(:,2));title('u2')                          30
subplot(2,2,3);plot(y(:,1));title('y1')
subplot(2,2,4);plot(y(:,2));title('y2')
% end of demo-2.
```

A.4.4 Demo-3 (Application To A Real System)

```
% SELF TUNING CONTROL OF REDUCED ORDER MODEL OF T700 TURBOSHAFT
% HELICOPTER ENGINE
clear
% References.
for i=1:100;yr(i,:)=700;end
for i=101:200;yr(i,:)=600;end
for i=201:300;yr(i,:)=650;end
for i=301:400;yr(i,:)=500;end
for i=401:500;yr(i,:)=550;end
% Disturbances.                                                    10
for i=1:50;e(i,:)=7;end
for i=51:150;e(i,:)=-6;end
for i=151:250;e(i,:)=6.5;end
for i=251:350;e(i,:)=-5;end
for i=351:450;e(i,:)=5.5;end
for i=451:500;e(i,:)=-7;end
%
% This part changes according to the system.
nu=1;ny=1;m=1;n=1;k=0;d=1;std=10;mean=0;RI=1E4;
W=0.97;QQ1=1;QQ2=1;umin=-0.05;umax=0.05;                           20
%
% This part should always be written, whatever system we use.
ne=ny;dmn=nu*(m+1)+n*ny+k*ne;
R=eye(dmn)*RI;Q1=eye(ny)*QQ1;Q2=eye(nu)*QQ2;
P=[eye(ny) zeros(ny,dmn-ny)];
ts=size(yr,1);t0=d+m+1;
if n>(d+m);t0=n+1;end
te=ts+t0-1;yr=[zeros(t0-1,ny);yr];y=zeros(te+d,ny);
u=zeros(te+d,nu);
e=[zeros(t0-1,ne);e];                                              30
%
%**** YOUR SPECIAL EXPRESSIONS FOR THE REAL SYSTEM ****
H=.02;
%       If State space equations are like these,
```

```
% .
% x=A.x+B.u
% y=C.x
%       for computing discrete model, we can take C=0. And
%       after obtaining discrete model equations, we can use our
%       real C matrix in the simulation.                              40
%
A=[-0.4474E+1 0.0 -0.6928E+3 0.1462E+4 -0.2942E+4;...
  -0.2444E-1 -0.5650 -0.4128E+2 0.3302E+2 0.2321E+3;...
  0.7524 0.0 -0.4230E+3 0.4071E+3 0.0;...
  0.0 0.0 0.4115E+4 -0.4526E+4 0.0;...
  0.1063E+1 0.0 -0.2695E+3 0.8101E+3 -0.3063E+4];
B=[0.8238E+5;0.6174E+4;0.0;0.1891E+6;0.4021E+5];
C=[0 0 0 0 0;0 0 0 0 0;0 0 0 0 0;0 0 0 0 0;0 0 0 0 0];
D=[0;0;0;0;0];
[AD,BD,CD,DD]=c2dm(A,B,C,D,H,'zoh');                                  50
% or [AD,BD,CD,DD]=c2dm(A,B,C,D,H,'tustin');
y(t0-1,:)=0;x(t0-1,:)=[0 0 0 0 0];% Initial condition matrix.
%*****************************************
for t=t0:te; % Loop should always be within these limits.
%***** YOUR REAL SYSTEM EQUATIONS ******
 x(t,:)=(AD*x(t-1,:)'+BD*u(t-d,:)')';
 yd(t,:)=x(t,1);% GAS TURBIN SPEED (NG)
 y(t,:)=x(t,2)+e(t,:);% POWER TURBIN SPEED (NP)
%*************************************
[R,P,u]=ucrstc(t,nu,ny,ne,m,n,k,yr,u,e,y,d,P,R,W,Q1,Q2,umin,umax);    60
end
u=u(t0-1:ts+t0-2,:);y=y(t0-1:ts+t0-2,:);yr=yr(t0-1:ts+t0-2,:);
%********* GRAPHS *********
subplot(2,2,1);plot(u);title('u')
subplot(2,2,2);plot(e);title('e')
subplot(2,2,3);plot([y(:,1) yr(:,1)]);title('NP')
subplot(2,2,4);plot(yd(:,1));title('NG')
%**************************
% end of demo-3.
```

A.4.5 Self-tuning Control Of A MIMO System (Demo-4)

% DEMO–5 ABOUT SELF TUNING CONTROL

% AUTHOR: Ali Yurdun ORBAK – Massachusetts Institute Of Technology

%

% A MIMO – Continuous Case

%

% The system has a transfer function as:

%

$$\% \; \mathbf{G}(s) = \frac{1}{(s^2+3\,s-1)} \begin{bmatrix} s+1 & 3 \\ 1 & s+2 \end{bmatrix}$$

%

%

```
clear;clf;
for i=1:500;yr(i,1:2)=[1 1.5];end
for i=1:500;e(i,1:2)=[0 0];end
disp('System:  MIMO        .')
disp('                     x=AC*x+BC*u')
disp('                     y=CC*x+DC*u')
AC=[-2 3;1 -1];
BC=[1 0;0 1];
CC=[1 0;0 1];
DC=[0 0;0 0];
delta=.015; % Time delay.
h=0.01;
d=fix(delta/h)+1;
[AD,BD,CD,DD]=c2dm(AC,BC,CC,DC,h,'zoh');
nu=2;ny=2;m=1;n=3;k=0;std=0;mean=0;RI=1E4;
W=0.98;QQ1=1;QQ2=0.002;umin=-8;umax=8;
ne=ny;dmn=nu*(m+1)+n*ny+k*ne;
R=eye(dmn)*RI;Q1=eye(ny)*QQ1;Q2=eye(nu)*QQ2;
P=[eye(ny) zeros(ny,dmn-ny)];
ts=size(yr,1);t0=d+m+1;
```

```
if n>(d+m);t0=n+1;end
te=ts+t0-1;yr=[zeros(t0-1,ny);yr];y=zeros(te+d,ny);
u=zeros(te+d,nu);
e=[zeros(t0-1,ne);e];
y(t0-1,:)=[0 0];x(t0-1,:)=[0 0];
for t=t0:te;
x(t,:)=(AD*x(t-1,:)'+BD*u(t-d,:)')';
y(t,1)=x(t,1)+e(t,1);
y(t,2)=x(t,2)+e(t,2);                                                    40
[R,P,u]=ucrstc(t,nu,ny,ne,m,n,k,yr,u,e,y,d,P,R,W,Q1,Q2,umin,umax);
end
u=u(t0-1:ts+t0-2,:);y=y(t0-1:ts+t0-2,:);yr=yr(t0-1:ts+t0-2,:);
subplot(2,2,1);plot(u(:,1));grid;title('Command Input u')
subplot(2,2,2);plot(u(:,2));grid;title('Command Input u')
subplot(2,2,3);plot(y(:,1));grid;title('Output y1')
subplot(2,2,4);plot(y(:,2));grid;title('Output y2')
end
%end of demo-5.
```

A.4.6 Identification Demonstration (Demo-5) (Simple System Identification Using Matlab^{TM})

```
%
% AUTHOR: Ali Yurdun ORBAK  -  Massachusetts Institute of Technology
%
% An .m file that makes parameter estimation and simulation for a
% system in the form:
%
% Y(s)/U(s) = e^(-delta*s) * (to1*s+1)/((to2*s+1)*(to3*s+1))
%
clear;clf;
to1=0.1;
to2=0.2;
to3=0.3;
delta=.015; % Time delay.
h=1; % Time interval.
W=0.98; % Weighing factor in RLS algorithm.
rs=1e4; % Starting value of matrix R.
kson=800;
us=1;
d=fix(delta/h)+1;
num=[to1 1];
den=[to2*to3 to2+to3 1];
disp('Y(s)/U(s)=')
printsys(num,den)
[A,B,C,D]=tf2ss(num,den); % Transfer function to state space.
[Aa,Bb,Cc,Dd]=c2dt(A,B,C,h,delta); % Continuous to discrete with delay.
% [numz,denz]=ss2tf(Aa,Bb,Cc,Dd);
% disp('Y(z)/U(z)=')
% printsys(numz,denz,'z')
%
[u,ub]=uinput(kson,us);
y=dlsim(Aa,Bb,Cc,Dd,u); % Simulation.;
```

```
plot(y);grid;title('Output (y)');pause;
y2=y;
u2=u;
z2=[y2(1:500) u2(1:500)];
idplot(z2,1:500,h);subplot(2,1,1);grid;subplot(2,1,2);grid;pause;
z2=dtrend(z2);
ir=cra(z2);subplot(2,2,1);grid;subplot(2,2,2);grid;
subplot(2,2,3);grid;subplot(2,2,4);grid;pause;                          40
stepr=cumsum(ir);
% subplot(1,1,1);plot(stepr);pause;
th=arx(z2,[2 2 1]);
th=sett(th,h);
present(th);
[numth,denth]=th2tf(th,1);
printsys(numth,denth)
u=dtrend(u2(500:800));
y=dtrend(y2(500:800));
yh=idsim(u,th);                                                         50
subplot(1,1,1);plot([yh y]);grid;
title('Comparison of Real Output vs.  Model Output');;pause;
step=ones(20,1);
mstepr=idsim(step,th);
plot([stepr mstepr]);grid;title('Comparison of Step Responses');pause
% idsimsd(step,th);pause
gth=th2ff(th);
gs=spa(z2);
gs=sett(gs,h);
bodeplot([gs gth]);subplot(2,1,1);grid;subplot(2,1,2);grid;             60
end
```

A.4.7 Identification Demonstration-2 (Demo-6)

```
%
% Author: Ali Yurdun Orbak – Massachusetts Institute of Technology
%
% An .m file that makes parameter estimation using identification toolbox
% of Matlab for the system that has a G(s) as:
%
% G(s) = 1/(s^2+35 s+250)
%
clear;clf;
delta=0; % Time delay.                                                    10
h=1; % Time interval.
W=0.98; % Weighing factor in RLS algorithm.
rs=1e4; % Starting value of matrix R.
kson=800;
us=1;
d=fix(delta/h)+1;
num=[1];
den=[1 35 250];
disp('Y(s)/U(s)=')
printsys(num,den)                                                         20
[A,B,C,D]=tf2ss(num,den); % Transfer function to state space.
[Aa,Bb,Cc,Dd]=c2dt(A,B,C,h,delta); % Continuous to discrete with delay.
% [numz,denz]=ss2tf(Aa,Bb,Cc,Dd);
% disp('Y(z)/U(z)=')
% printsys(numz,denz,'z')
%
[u,ub]=uinput(kson,us);
y=dlsim(Aa,Bb,Cc,Dd,u); % Simulation.;
plot(y);grid;title('Output (y)');pause;
y2=y;                                                                     30
u2=u;
z2=[y2(1:500) u2(1:500)];
idplot(z2,1:500,h);subplot(2,1,1);grid;subplot(2,1,2);grid;pause;
z2=dtrend(z2);
```

```
ir=cra(z2);subplot(2,2,1);grid;subplot(2,2,2);grid;
subplot(2,2,3);grid;subplot(2,2,4);grid;pause;
stepr=cumsum(ir);
% subplot(1,1,1);plot(stepr);pause;
th=arx(z2,[1 2 0]);
th=sett(th,h);                                                    40
present(th);
[numth,denth]=th2tf(th,1);
printsys(numth,denth)
u=dtrend(u2(500:800));
y=dtrend(y2(500:800));
yh=idsim(u,th);
subplot(1,1,1);plot([yh y]);grid;
title('Comparison of Real Output vs.  Model Output');;pause;
step=ones(20,1);
mstepr=idsim(step,th);                                            50
plot([stepr mstepr]);grid;title('Comparison of Step Responses');pause
% idsimsd(step,th);pause
gth=th2ff(th);
gs=spa(z2);
gs=sett(gs,h);
bodeplot([gs gth]);subplot(2,1,1);grid;subplot(2,1,2);grid;
end
```

A.5 General System Identification Programs

```
function [P,yd,V]=lsmiden(y,u,m,n,d)
% IDENTIFICATION OF MIMO SYSTEMS
%   USING LEAST SQUARES METHOD
%
%
% [P,yd,V]=lsmiden(y,u,m,n,d)
%
% y(z^-1)   z^-d*(B0+B1*z^-1+B2*z^-2+..+Bm*z^-m)
% -------- = ------------------------------------
% u(z^-1)       I+A1*z^-1+A2*z^-2+..+An*z^-n                    10
%
%   |y(1)|    |u(1)|
%   |y(2)|    |u(2)|    y(t)=[y1(t) y2(t) .. yny(t)]
% y=| : |   u=| : |
%   | : |    | : |    u(t)=[u1(t) u2(t) .. unu(t)]
%   |y(t)|    |u(t)|
%
%                    yd : Calculated output data
%
% P=[A B]=[A1 A2 .. An B0 B1 .. Bm]                              20
%
ny=size(y,2);
nu=size(u,2);
p1=1;
if ny>2;
p1=2;
end
p2=ny;
if p1==2;
p2=fix((ny+1)/2);                                              30
end
%
P=lstsqr(y,u,m,n,d);
[yd,V]=arxsim(P,u,y,m,n,d);
```

```
for i=1:ny;
subplot(p2,p1,i);
plot([yd(:,i) y(:,i)])
title(['y' setstr(48+i)])
end
end                                                    40
% End of file
```

A.5.1 Identification Function

```
function [yd,V]=arxsim(P,u,y,m,n,d)
% MIMO ARX MODEL SIMULATION
%
%
% [yd,V]=arxsim(P,u,y,m,n,d)
%
% yd: Calculated Output Data
% V: Loss Function
% P: Parameter matrix
% u: Input data                                                    10
% y: Output data
% m: B polynomial order
% n: A      "       "
% d: delay
%
ny=size(y,2);
ts=size(y,1);
t0=d+m+1;
if n>(m+d);t0=n+1;end
for i=1:t0-1;                                                    20
yd(i,:)=y(i,:);
end
%
for t=t0:ts;
fi=-yd(t-1,:)';
for j=2:n;fi=[fi;-yd(t-j,:)'];end
for i=0:m;fi=[fi;u(t-d-i,:)'];end
yd(t,:)=(P*fi)';
end
V=funcloss(y,yd);                                                30
%
end
```

A.5.2 Loss Function Calculation

```
% Calculating Loss Function

function [loss]=funcloss(y,yc)
loss=0;
for t=1:size(y,1);
V=y(t,:)-yc(t,:);
loss=loss+V*V';
end
```

A.5.3 General Least Squares Estimation Function

```
function P=lstsqr(y,u,m,n,d)
% LEAST SQUARES PARAMETER ESTIMATION
%
% P=lstsqr(y,u,m,n,d)
% y(z^-1)   z^-d*(B0+B1*z^-1+B2*z^-2+..+Bm*z^-m)
% ------- = -------------------------
% u(z^-1)       I+A1*z^-1+A2*z^-2+..+An*z^-n
%   |y(1)|    |u(1)|
%   |y(2)|    |u(2)|    y(t)=[y1(t) y2(t) .. yny(t)]
%  y=| : |   u=| : |                                          10
%   | : |    | : |    u(t)=[u1(t) u2(t) .. unu(t)]
%   |y(t)|    |u(s)|
%
% P=[A B]=[A1 A2 .. An B0 B1 .. Bm]
%
ts=size(y,1);
er=0;
if (d+m-n+1)<1;
er=d-m+n;
end                                                          20
ym=y(d+m+1+er:ts,:);
fi=[];
for i=1:n;
fi=[fi -y(d+m+1-i+er:ts-i,:)];
end
for j=0:m;
fi=[fi u(m+1-j+er:ts-d-j,:)];
end
P=((fi'*fi)\(fi'*ym))';  % Actually P=(inv(fi'*fi)*fi'*ym)' .
                         % When det(fi'*fi)=0, P becomes wrong  30
                         % with this expression.
end
% end of function
```

A.5.4 System Identification Using Lsmiden.m file

```
% Author: Ali Yurdun Orbak – Massachusetts Institute of Technology
% An .m file that makes parameter estimation using lsmiden.m file
% for the system that has a G(s) as:
%
% G(s) = 1/(9.375 s^2+6.25 s+1)
clear
num=[1];
den=[9.375 6.25 1];
t(1)=0;
kson=300;
us=1;
[u,ub]=uinput(kson,us);
p=size(u);
k=p(1,1);
for i=1:k-1
t(i+1)=t(i)+1;
end
y=lsim(num,den,u,t);
m=2;n=2;d=0;
[P,yd,V]=lsmiden(y,u,m,n,d);
clf
plot(yd,'go')
hold on
plot(y)
legend('Model','Real')
yep=lsim(num,den,ub,t);
figure(2)
plot(yep(1:30));
hold on
num1=[P(1,3) P(1,4) P(1,5)];
den1=[1 P(1,1) P(1,2)];
yep1=dlsim(num1,den1,ub);
plot(yep1(1:30),'go');
```

A.5.5 System Identification and Self-tuning Control of A SIMO System

```
% DEMO-6 ABOUT SELF TUNING CONTROL
% AUTHOR: Ali Yurdun ORBAK - Massachusetts Institute Of Technology
%
% A MIMO - Continuous Case  Example Using Lsmiden.m (ARX modelling) file -
% Single Input Multi Output (One Input Two Outputs)
%
%
clear;clf;
n1=[0 1];n2=[1 1];
num=[n1;n2];                                                          10
den=[1 5 6];
disp('System:  SIMO        .')
disp('                     x=AC*x+BC*u')
disp('                     y=CC*x+DC*u')
[AC,BC,CC,DC]=tf2ss(num,den)
delta=.015; % Time delay.
h=0.01;
kson=500;
us=1;
d=fix(delta/h)+1;                                                     20
[AD,BD,CD,DD]=c2dt(AC,BC,CC,h,delta);
% or [AD,BD,CD,DD]=c2dm(AC,BC,CC,DC,h,'zoh');
[numz,denz]=ss2tf(AD,BD,CD,DD);
[u,ub]=uinput(kson,us);
y=dlsim(AD,BD,CD,DD,u); % Simulation.
m=1;n=3;
[P,yd,V]=lsmiden(y,u,m,n,d);
A1=P(:,1:2);
A2=P(:,3:4);
A3=P(:,5:6);                                                          30
B0=P(:,7);
B1=P(:,8);
```

```
AR=[eye(2) A1 A2 A3];
BR=[B0 B1];
CR=eye(2);
for i=1:500;yr(i,1:2)=[1 1];end
y0=[0 0];
nu=1;
RI=1E+04;
W=0.98;                                                          40
QQ1=1;
QQ2=0.001;
std=0.001;mean=0;
umin=-35;umax=35;
[u,e,y,P]=genstc('tm',AR,BR,CR,nu,y0,yr,d,std,mean,RI,W,QQ1,QQ2,umin,umax);
subplot(2,2,1);plot(u(:,1));grid;title('Command Input u')
subplot(2,2,3);plot(y(:,1));grid;title('Output y1')
subplot(2,2,4);plot(y(:,2));grid;title('Output y2')
end
%end of demo-6.                                                   50
```

Appendix B

About Modelling

The identification part of dynamic system modelling is usually the most crucial part. Lack of appropriate identification, the modelling and control parts will be useless.

The general method of identification can be summarized by figure B-1. Here, the choice of identification criterion and technique are chosen according to the problem and a priori knowledge in hand.

Now let's list the common identification techniques.

B.1 ARX Modelling

Consider that it is desired to estimate the parameters a_i and b_i of the transfer function:

$$G(z) = \frac{b_0\, z^n + b_1\, z^{n-1} + \cdots + b_n}{z^n + a_1\, z^{n-1} + \cdots + a_n} \tag{B.1}$$

from input and output data $u(t)$ and $y(t)$. Assuming that the data and/or the model are not exact, then the following is a commonly used model for this input-output relation:

$$\begin{aligned} y(t) &= -\sum_{i=1}^{n} a_i\, y(t-i) + \sum_{i=0}^{n} b_i\, u(t-i) + v(t) \\ &= \boldsymbol{\theta}^T \boldsymbol{\varphi}(t) + v(t) \end{aligned} \tag{B.2}$$

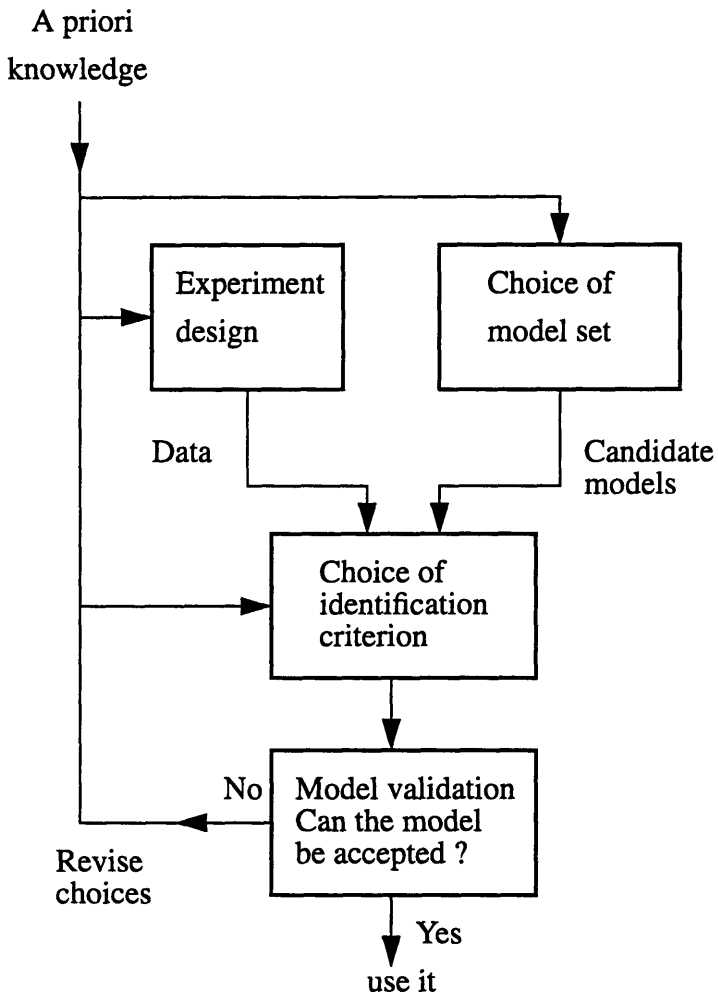

Figure B-1: Summary of Identification Procedure

where

$$\varphi(t) = (-y(t-1) \cdots -y(t-n)\ \ u(t) \cdots u(t-n))^T \tag{B.3}$$

is the regressor vector, and

$$\boldsymbol{\theta} = \begin{pmatrix} \mathbf{a} \\ \bar{\mathbf{b}} \end{pmatrix} \tag{B.4}$$

is the parameter vector, $\mathbf{a}$ and $\bar{\mathbf{b}}$ are defined as

$$\left.\begin{aligned} \mathbf{a} &= (a_1 \cdots a_n)^T \\ \bar{\mathbf{b}} &= (b_0\, b_1 \cdots b_n)^T \end{aligned}\right\} \tag{B.5}$$

and $v(t)$ is a noise process.

The model(equation (B.2)) is an equation error model, because an error $v(t)$ is assumed on the input-output equation; it is more commonly called an ARX[1] model or a Least Squares regression model in identification language.[2]

B.2 Common Identification Algorithms

Here is a list of most commonly used identification techniques.

a) *UD Factorization Methods*

$$\mathbf{P}(t) = \mathbf{U}(t)\,\mathbf{D}(t)\,\mathbf{U}^T(t) \tag{B.6}$$

where $\mathbf{U}(t)$ is an upper triangular matrix with all diagonal elements equal to 1, and where $\mathbf{D}(t)$ is a diagonal matrix[3]. More discussion about this identification can be found in [31].

b) *Ladder and Lattice Algorithms*

c) *Weighted Least Squares*

d) *Colored Equation Error Noise*

e) *Extended Least Squares*

f) *Maximum Likelihood Method*

g) *Grey-Box Model Identification*

h) *Identification using Bond Graphs*

i) *Gradient Methods*

---

[1]This name was given by econometricians, and supposed to mean **A**uto **R**egressive with e**X**ogeneous inputs.

[2]Note that it is model in the shift operator, because the regressor contains delayed (i.e. shifted) versions of the input and output signals.

[3]The resulting $\mathbf{P}(t)$ is the same that was used in the Recursive Parameter estimation part of this thesis.

Appendix C

Proof of the Matrix Inversion Lemma

This appendix was taken from [32] to help those who wishes to reduce the computational load of their algorithms.

In a number of problems of optimal control, state estimation, and parameter estimation, one is faced with the task of computing the inverse of a composite matrix to give its inverse $\mathbf{M}$. The matrix $\mathbf{M}$ to be calculated is of a particular form, which is

$$\mathbf{M} = (\mathbf{P}^{-1} + \mathbf{B}\,\mathbf{R}^{-1}\,\mathbf{B}^{T})^{-1}$$

where $\mathbf{P}$ and $\mathbf{R}$ are such that $\mathbf{P}$ is $n \times n$, $\mathbf{R}$ is $m \times m$, and $\mathbf{B}$ is $n \times m$. In most problems of interest m is less than n. Let us evaluate $\mathbf{M}^{-1}$ or

$$\mathbf{M}^{-1} = \mathbf{P}^{-1} + \mathbf{B}\,\mathbf{R}^{-1}\,\mathbf{B}^{T} \tag{C.1}$$

Now premultiply by $\mathbf{M}$ to give

$$\mathbf{I} = \mathbf{M}\,\mathbf{P}^{-1} + \mathbf{M}\,\mathbf{B}\,\mathbf{R}^{-1}\,\mathbf{B}^{T}$$

Now postmultiply by $\mathbf{P}$ to give

$$\mathbf{P} = \mathbf{M} + \mathbf{M}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}$$

Now postmultiply by $\mathbf{B}$ and factor the product $\mathbf{M}\mathbf{B}$ out of the left to give

$$\mathbf{P}\mathbf{B} = \mathbf{M}\mathbf{B}(\mathbf{I} + \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}\mathbf{B})$$

Factor $\mathbf{R}^{-1}$ noting that $\mathbf{I} = \mathbf{R}^{-1}\mathbf{R}$; then the result is

$$\mathbf{P}\mathbf{B} = \mathbf{M}\mathbf{B}\mathbf{R}^{-1}(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})$$

Postmultiply by the inverse of the matrix in parentheses:

$$\mathbf{P}\mathbf{B}(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})^{-1} = \mathbf{M}\mathbf{B}\mathbf{R}^{-1}$$

Now postmultiply by $\mathbf{B}^T$ to give

$$\mathbf{P}\mathbf{B}(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^T = \mathbf{M}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T$$

The matrix $\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T$ may be replaced by $\mathbf{M}^{-1} - \mathbf{P}^{-1}$ from relation (B.1) to yield

$$\mathbf{P}\mathbf{B}(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^T = \mathbf{M}(\mathbf{M}^{-1} - \mathbf{P}^{-1})$$

Recognizing that $\mathbf{M}\mathbf{M}^{-1}$ is the identity matrix gives

$$\mathbf{P}\mathbf{B}(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^T = \mathbf{I} - \mathbf{M}\mathbf{P}^{-1}$$

Now postmultiply by $\mathbf{P}$ to give

$$\mathbf{P}\mathbf{B}(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{P} = \mathbf{P} - \mathbf{M}$$

The solution for $\mathbf{M}$ is thus

$$\mathbf{M} = \mathbf{P} - \mathbf{P}\,\mathbf{B}\,(\mathbf{R} + \mathbf{B}^T\,\mathbf{P}\,\mathbf{B})^{-1}\,\mathbf{B}^T\,\mathbf{P}$$

Now we see that the only matrix which must be inverted is $\mathbf{R} + \mathbf{B}^T\,\mathbf{P}\,\mathbf{B}$, which is $m \times m$, and hence smaller than the original $n \times n$ matrix. In the case where $m = 1$, which is not uncommon, $\mathbf{R}$ is scalar and hence the inverse is given by a simple division.

Bibliography

[1] Winfred K. N. Anakwa and Jinn-Huei Lan. MOTOROLA DSP56001 implementation of a self-tuning controller. In *Proceedings of the American Control Conference*, pages 2241–2245. American Automatic Control Council, June 1993.

[2] K. J. Åström. Self-tuning regulators-design principles and applications. In Kumpati S. Narenda and Richard V. Monopoli, editors, *Applications of Adaptive Control*, pages 1–68. Academic Press Inc., New York, 1980.

[3] K. J. Åström. Directions in intelligent control. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 1–9. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[4] K. J. Åström, U. Borisson, L. Ljung, and B. Wittenmark. Theory and applications of self-tuning regulators. *Automatica*, 13:457–476, 1977.

[5] K. J. Åström and B. Wittenmark. On self-tuning regulators. In Madan M. Gupta and Chi-Hau Chen, editors, *Adaptive Methods for Control System Design*, pages 181–194. IEEE Press, New York, 1986.

[6] Karl Johan Åström and Björn Wittenmark. *Adaptive Control.* Addison-Wesley, Reading, Massachusetts, first edition, 1989.

[7] M. G. Ballin. A high fidelity real time simulation of a small turboshaft engine. Technical Report TM 100991, NASA, July 1988.

[8] E. A. Barros and H. M. Morishita. Dynamic positioning system using self-tuning control. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*,

number 7 in IFAC Symposia Series, pages 307–312. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[9] J. Böhm and M. Kárný. Merging of user's knowledge into identification part of self-tuners. In L. Dugard, M. M'saad, and I. D. Landau, editors, *Adaptive Systems in Control and Signal Processing*, number 8 in IFAC Symposia Series, pages 273–278. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1993.

[10] Ulf Borison and Rolf Syding. Self-tuning control of an ore-crusher. In Madan M. Gupta and Chi-Hau Chen, editors, *Adaptive Methods for Control System Design*, pages 388–394. IEEE Press, New York, 1986.

[11] Han-Fu Chen and Ji-Feng Zhang. Identification and adaptive control for systems with unknown orders, delay, and coefficients. *IEEE Transactions on Automatic Control*, 35(8):866–877, August 1990.

[12] D. W. Clarke. Implementation of self-tuning controllers. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control: Theory and Applications*, number 15 in IEE Control Engineering Series, chapter 6, pages 144–165. Peter Peregrinus Ltd., London and New York, 1981.

[13] D. W. Clarke. Introduction to self-tuning controllers. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control: Theory and Applications*, number 15 in IEE Control Engineering Series, chapter 2, pages 36–71. Peter Peregrinus Ltd., London and New York, 1981.

[14] D. W. Clarke. Introduction to self-tuning control. In Kewin Warwick, editor, *Implementations of Self-tuning Controllers*, number 35 in IEE Control Engineering Series, chapter 1, pages 1–22. Peter Peregrinus Ltd., London, United Kingdom, 1988.

[15] D. W. Clarke, D. Phil, and P. J. Gawthrop. Self-tuning control. In Madan M. Gupta and Chi-Hau Chen, editors, *Adaptive Methods for Control System Design*, pages 195–202. IEEE Press, New York, 1986.

[16] John P. Clary and Gene F. Franklin. Self-tuning control with a priori plant knowledge. *IEEE Proceedings of 23rd Conference on Decision and Control*, pages 369–374, December 1984.

[17] Premal Desai and A. K. Mahalanabis. A state space self-tuning controller with integral action. *Optimal Control Applications and Methods*, 13:301–319, 1992.

[18] R. Devanathan. Expert self-tuning PI(D) controller. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 217–222. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[19] R. Devanathan. A self-tuning feedforward compensator. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 393–398. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[20] Zhengtao Ding and Brian W. Hogg. Robust self-tuning control through detecting changes in system dynamics. In *Proceedings of the 32nd Conference on Decision and Control*, pages 1589–1590. IEEE Control System Society, IEE Conference Publication, December 1993.

[21] Guy A. Dumont and Pierre R. Bélanger. Self-tuning control of a titanium dioxide kiln. In Madan M. Gupta and Chi-Hau Chen, editors, *Adaptive Methods for Control System Design*, pages 395–401. IEEE Press, New York, 1986.

[22] A. L. Elshafei, G. A. Dumont, and Ye Fu. Orthonormal functions in identification and adaptive control. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 193–198. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[23] P. T. K. Fung and M. J. Grimble. Self-tuning control for ship positioning systems. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control: Theory and Applications*, number 15 in IEE Control Engineering Series, chapter 14, pages 309–331. Peter Peregrinus Ltd., London and New York, 1981.

[24] P. J. Gawthrop. On the stability and convergence of a self-tuning controller. *International Journal of Control*, 31(5):973–998, 1980.

[25] P. J. Gawthrop and R.W. Jones. Bond-graph based adaptive control. In L. Dugard, M. M'saad, and I. D. Landau, editors, *Adaptive Systems in Control and Signal Processing*, number 8 in IFAC Symposia Series, pages 67–72. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1993.

[26] M. J. Grimble. H_∞ robust controller for self-tuning control applications. part 1. controller design. *International Journal of Control*, 46(4):1429–1444, 1987.

[27] M. J. Grimble. H_∞ robust controller for self-tuning control applications. part 2. self-tuning and robustness. *International Journal of Control*, 46(5):1819–1840, 1987.

[28] C. C. Hang and D. Chin. Reduced order process modeling in self-tuning control. *Automatica*, 27(3):529–534, 1991.

[29] Miklós Hilger, László Keviczky, Jenö Hetthéssy, and János Kolostori. Self-tuning adaptive control of cement raw material blending. In Madan M. Gupta and Chi-Hau Chen, editors, *Adaptive Methods for Control System Design*, pages 402–409. IEEE Press, New York, 1986.

[30] Håkan Hjalmarsson and Lennart Ljung. Estimating model variance in the case of undermodeling. *IEEE Transactions on Automatic Control*, 37(7):1004–1008, July 1992.

[31] Editor in chief: Madan G. Singh. *Systems and Control Encyclopedia. Theory, Technology, Applications.* Eight volumes. Pergamon Press, Oxford, New York, Beijing, Frankfurt, São Paolo, Sydney, Tokyo, Toronto, first edition, 1987.

[32] Raymond G. Jacquot. *Modern Digital Control Systems*, chapter 13, pages 367–383. Marcel Dekker, Inc., Newyork, Basel, Hong Kong, second edition, 1995.

[33] Raymond G. Jacquot. *Modern Digital Control Systems*, appendix C. Marcel Dekker, Inc., Newyork, Basel, Hong Kong, second edition, 1995.

[34] H. N. Koivo and J. T. Tanttu. Tuning of PID controllers: Survey of SISO and MIMO techniques. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 75–80. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[35] Robert L. Kosut, Ming Lau, and Stephen Boyd. Identification of systems with parametric and nonparametric uncertainty. In *Proceedings of the 1990 American Control Conference*, pages 2412–2417, May 1990.

[36] A. J. Krijgsman, H. B. Verbruggen, and P. M. Bruijn. Knowledge-based tuning and control. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 411–416. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[37] P. R. Kumar. Convergence of adaptive control schemes using least-squares parameter estimates. *IEEE Transactions on Automatic Control*, 35(4):416–424, April 1990.

[38] K. W. Lim and S. Ginting. A non-minimal model for self-tuning control. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 133–137. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[39] P. Lin, Y. S. Yun, J. P. Barbier, Ph. Babey, and P. Prevot. Intelligent tuning and adaptive control for cement raw blending process. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 301–306. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[40] L. Ljung. Model accuracy in system identification. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 277–281. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[41] Lennart Ljung. On positive real transfer functions and the convergence of some recursive schemes. *IEEE Transactions on Automatic Control*, 22(4):539–551, August 1977.

[42] M. Molander, P. E. Modén, and K. Holmström. Model reduction in recursive least squares identification. In L. Dugard, M. M'saad, and I. D. Landau, editors, *Adaptive Systems in Control and Signal Processing*, number 8 in IFAC Symposia Series, pages 5–10. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1993.

[43] A. J. Morris, Y. Nazer, and R. K. Wood. Single and multivariable application of self-tuning controllers. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control: Theory and Applications*, number 15 in IEE Control Engineering Series, chapter 11, pages 249–281. Peter Peregrinus Ltd., London and New York, 1981.

[44] N. Mort and D. A. Linkens. Self-tuning controllers for surface ship course-keeping and manoeuvring. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control: Theory and Applications*, number 15 in IEE Control Engineering Series, chapter 13, pages 296–308. Peter Peregrinus Ltd., London and New York, 1981.

[45] P. A. J. Nagy and L. Ljung. System identification using bondgraphs. In L. Dugard, M. M'saad, and I. D. Landau, editors, *Adaptive Systems in Control and Signal Processing*, number 8 in IFAC Symposia Series, pages 61–66. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1993.

[46] V. Paterka. Predictor based self-tuning control. In Madan M. Gupta and Chi-Hau Chen, editors, *Adaptive Methods for Control System Design*, pages 231–242. IEEE Press, New York, 1986.

[47] William H. Pfeil, Michael Athans, and H. Austin Spang, III. Multivariable control of the GE T700 engine using the LQG/LTR design methodology. In *Proceedings of the American Control Conference*, pages 1297–1312. American Automatic Control Council, 1986.

[48] A. Basharati Rad and P. J. Gawthrop. Explicit PID self-tuning control for systems with unknown time delay. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 251–257. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[49] J. C. Readle and R. M. Henry. On-line determination of time-delay using multiple recursive estimators and fuzzy reasoning. In *Control'94*, pages 1436–1441. IEE, IEE Conference Publication, March 1994.

[50] Hideaki Sakai. Generalized predictive self-tuning control for tracking a periodic reference signal. *Optimal Control Applications and Methods*, 13:321–333, 1992.

[51] Bahram Shahian and Michael Hassul. *Control System Design Using Matlab.* Prentice Hall,Inc., Englewood Cliffs, New Jersey, 1993.

[52] Kenneth R. Shouse and David G. Taylor. A digital self-tuning tracking controller for permanent-magnet synchronous motors. In *Proceedings of the 32nd Conference on Decision and Control*, pages 3397–3402. IEEE Control System Society, IEE Conference Publication, December 1993.

[53] Roy S. Smith and John C. Doyle. Model validation: A connection between robust control and identification. *IEEE Transactions on Automatic Control*, 37(7):942–952, July 1992.

[54] M. O. Tade, M. M. Bayoumi, and D. W. Bacon. Self-tuning controller design for systems with arbitrary time delays. part 1. theoretical development. *International Journal of Systems*, 19(7):1095–1115, 1988.

[55] M. O. Tade, M. M. Bayoumi, and D. W. Bacon. Self-tuning controller design for systems with arbitrary time delays. part 2. algorithms and simulation examples. *International Journal of Systems*, 19(7):1117–1141, 1988.

[56] M. Tadjine, M. M'saad, and M. Bouslimani. Self-tuning partial state reference model controller with loop transfer recovery. In *Control'94*, pages 777–782. IEE, IEE Conference Publication, March 1994.

[57] H. Takatsu, T. Kawano, and K. Kitano. Intelligent self-tuning PID controller. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 11–15. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[58] P. H. Thoa, N. T. Loan, and H. H. Son. Self-tuning adaptive control based on a new parameter estimation method. In R. Devanathan, editor, *Intelligent Tuning and Adaptive Control*, number 7 in IFAC Symposia Series, pages 345–350. Pergamon Press, Oxford, New York, Seoul, Tokyo, 1991.

[59] M. O. Tokhi, A. A. Hossain, and K. Mamour. Self-tuning active control of noise and vibration. In *Control'94*, pages 771–776. IEE, IEE Conference Publication, March 1994.

[60] Vance J. Vandoren. The challenges of self-tuning control. *Control Engineering*, pages 77–79, February 1994.

[61] P. E. Wellstead and D. L. Prager. Self-tuning multivariable regulators. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control: Theory and Applications*, number 15 in IEE Control Engineering Series, chapter 3, pages 72–92. Peter Peregrinus Ltd., London and New York, 1981.

[62] P. E. Wellstead and P. M. Zanker. Application of self-tuning to engine control. In C. J. Harris and S. A. Billings, editors, *Self-tuning and Adaptive Control: Theory and Applications*, number 15 in IEE Control Engineering Series, chapter 12, pages 282–295. Peter Peregrinus Ltd., London and New York, 1981.

[63] P. E. Wellstead and M. B. Zarrop. *Self-tuning Systems. Control and signal processing.* John Wiley and Sons, Chichester, New York, Brisbane, Toronto, Singapore, 1991.

[64] Björn Wittenmark and Karl Johan Åström. Practical issues in the implementation of self-tuning control. In Madan M. Gupta and Chi-Hau Chen, editors,

Adaptive Methods for Control System Design, pages 243–253. IEEE Press, New York, 1986.

[65] Takashi Yahagi and Jianming Lu. On self-tuning control of nonminimum phase discrete systems using approximate inverse systems. *Journal of Dynamic Systems, Measurement, and Control*, 115:12–18, March 1993.

[66] Tae-Woong Yoon and David W. Clarke. Towards robust adaptive predictive control. In David Clarke, editor, *Advances in Model-based Predictive Control*, pages 402–414. Oxford University Press, Oxford, New York, Tokyo, 1994.

[67] A. M. Zikic. *Practical Digital Control.* Ellis Horwood Series in Electrical and Electronic Engineering. Ellis Horwood Limited, Publishers, Chichester, first edition, 1989.

Biography

Mr. Âli Yurdun Orbak was born in İstanbul-Turkey in 1970. He graduated from 50.Yıl Cumhuriyet Elementary School. After five years of education, he took the highly competitive High School Entrance Exam and was admitted to İstanbul Kadıköy Anadolu High School which is one of the best high schools in Turkey. English and German are his first and second foreign languages respectively. He graduated from high school in 1988. At the end of high school, he took the university entrance exam with about seven hundred thousand candidates and was accepted to İstanbul Technical University's Mechanical Engineering department which accepts only students in the first percentile. He completed his undergraduate studies with the degree of S.B. in July 1992 with a rank of 1 out of 178 graduating students.

After graduating and ranking first from his faculty, he enrolled in an S.M. program in Robotics at the İstanbul Technical University Institute of Science & Technology. During his studies, he took the Turkish National Exam for a (masters and doctorate) graduate scholarship which was arranged by the Turkish Ministry of National Education, and was awarded a scholarship by the Turkish Government in order to pursue graduate studies in the US in Mechanical Engineering-Robotics. After applying to the best graduate schools in the US, he was accepted to MIT for Spring'94.

His research interests generally lie in interdisciplinary areas such as Robotics and Cybernetics. During his undergraduate studies, he was principally concerned with computer simulation, numerical analysis, control techniques and digital control systems. His S.B. thesis was about "Friction Compensation in DC Motor Drives" and especially on *A Position Sensor Based Torque Control Method for a DC Motor with Reduction Gears.* In this study his aim was to make a DC motor behave like a pure inertia, independent of the load. For this aim, a computer program was developed so as to find out the characteristics of the friction and to estimate necessary gains and other factors in order to compensate this friction. In that study, he also used a Japanese made HT-8000 DC motor with reduction gears and compared the results of the simulation with the real system.